

LUCAS EGA ACADEMY



HTML5

MODERN WEB DEVELOPMENT • RESPONSIVE
INTERACTIVE • PERFORMANCE



Curso Completo de HTML5

© 2026 Esteban Gutiérrez

Colibrí Editorial EGA

Hecho en México

lucasegaacademy.com

Curso Completo de HTML5

Autor: Esteban Gutiérrez

lucasegaacademy.com

*Guía completa con explicaciones extensas por unidad,
prácticas y examen final.*

ÍNDICE

PRÓLOGO	4
UNIDAD 1: Introducción a HTML5 y primeros pasos	9
UNIDAD 2: Texto, párrafos y navegación	43
UNIDAD 3: Listas, tablas y contenedores	80
UNIDAD 4: Atributos y formularios	122
UNIDAD 5: Validación, semántica y SEO	161
UNIDAD 6: Multimedia y tablas profesionales	204
UNIDAD 7: Gráficos y listas especializadas	239
UNIDAD 8: Precisión, accesibilidad y buenas prácticas	278
UNIDAD 9: Formularios profesionales y estructura del proyecto	312
UNIDAD 10: Proyecto final y graduación	350
EXAMEN FINAL	382

PRÓLOGO

Bienvenido a "Curso Completo de HTML5", el libro oficial del curso gratuito de Lucas EGA Academy. Si estás leyendo estas líneas es porque decidiste dar el primer paso en el mundo del desarrollo web, y déjame decirte algo desde el inicio: tomaste una gran decisión.

HTML5 es el lenguaje con el que están construidas todas las páginas web del planeta. Cada sitio que visitas, cada tienda en línea donde compras, cada red social donde pasas horas, todo, absolutamente todo, tiene HTML5 en su base. Es el esqueleto de la web. Y lo mejor de todo es que es el punto de partida perfecto: no necesitas saber programar, no necesitas matemáticas avanzadas, no necesitas una computadora cara. Solo necesitas ganas de aprender y las ganas ya las traes, porque aquí estás.

Este libro nació del curso en video de 30 clases de Lucas EGA Academy, que puedes tomar gratis en lucasegaacademy.com. Pero no es una simple transcripción. Aquí cada tema se explica con mucha más profundidad: con analogías de la vida real para que los conceptos se te queden grabados, con los errores más comunes que cometen los principiantes (y cómo evitarlos), con buenas prácticas profesionales, con ejemplos paso a paso comentados línea por línea, y con prácticas con soluciones explicadas para que compruebes que de verdad dominas cada tema.

Para quién es este libro

Este libro es para ti si estás en alguno de estos casos:

- Nunca has escrito una línea de código en tu vida y quieres empezar desde cero, sin suponer nada.
- Ya viste algunos tutoriales sueltos en internet pero sientes que te faltan las bases sólidas.
- Estudias o trabajas en algo relacionado con tecnología y necesitas entender cómo funciona la web por dentro.
- Quieres cambiar de carrera y el desarrollo web te llama la atención como una opción real de trabajo.
- Eres curioso por naturaleza y quieres construir tus propias páginas web, tu portafolio, tu blog, tu tienda.

No importa tu edad ni a qué te dediques. He visto estudiantes de 15 años y de 60 aprender HTML5 con este método. La única condición es que practiques: leer sin practicar es como

aprender a nadar leyendo un manual sin meterte al agua. En cada unidad hay ejercicios con soluciones, úsalos.

Cómo está organizado este libro

El libro está dividido en 10 unidades, y cada unidad agrupa 3 clases del curso en video. En total cubrimos las 30 clases completas:

- UNIDAD 1: Introducción a HTML5 y primeros pasos. Qué es HTML5, la estructura básica de un documento y las etiquetas de texto fundamentales.
- UNIDAD 2: Texto, párrafos y navegación. Cómo organizar texto con párrafos, crear enlaces que conectan la web y mostrar imágenes.
- UNIDAD 3: Listas, tablas y contenedores. Listas ordenadas y no ordenadas, tablas para datos tabulares, y los contenedores div y span.
- UNIDAD 4: Atributos y formularios. Los atributos globales id y class, y tus primeros formularios con los inputs inteligentes de HTML5.
- UNIDAD 5: Validación, semántica y SEO. Validación nativa sin JavaScript, etiquetas semánticas que Google ama, y meta tags para posicionar.
- UNIDAD 6: Multimedia y tablas profesionales. Audio y video nativos, tablas avanzadas con thead/tbody/tfoot, e iframes para incrustar contenido.
- UNIDAD 7: Gráficos y listas especializadas. Canvas para dibujar con código, gráficos vectoriales SVG y listas de definición.
- UNIDAD 8: Precisión, accesibilidad y buenas prácticas. Entidades HTML, accesibilidad web para todos, y los hábitos de un profesional.
- UNIDAD 9: Formularios profesionales y estructura del proyecto. Fieldset, datalist, etiquetas obsoletas que debes evitar, y la base de tu portafolio.
- UNIDAD 10: Proyecto final y graduación. Las secciones de tu portafolio, el formulario de contacto, repaso general y tu siguiente paso: CSS3.

Cada unidad sigue la misma estructura para que siempre sepas qué esperar: una introducción, la explicación detallada con analogías y errores comunes, ejemplos paso a paso con código comentado, una sección de práctica con ejercicios resueltos, y un resumen final. Al terminar las 10 unidades te espera el EXAMEN FINAL de 50 preguntas con todas sus respuestas explicadas.

Cómo estudiar este libro

Te propongo un método sencillo que ha funcionado con miles de estudiantes:

Primero, lee la unidad completa de corrido, sin prisa y sin tratar de memorizar todo. El objetivo de la primera lectura es entender la idea general: qué es cada etiqueta, para qué sirve, cómo encaja con lo demás. Subraya o anota lo que no te quede claro.

Segundo, vuelve a la sección de ejemplos paso a paso y escribe el código tú mismo en tu computadora. No lo copies y pegues: escríbelo letra por letra. Tus dedos también aprenden. Abre el archivo en tu navegador y ve el resultado. Luego rompe el código a propósito: quita una etiqueta de cierre, cambia un atributo, y observa qué pasa. Equivocarte a propósito es una de las formas más rápidas de aprender.

Tercero, haz los ejercicios de la sección de práctica sin mirar las soluciones. Date permiso de batallar un rato con cada ejercicio; esa lucha es donde ocurre el aprendizaje de verdad. Solo cuando tengas tu intento (aunque esté incompleto), compara con la solución explicada y entiende cada diferencia.

Cuarto, al terminar cada unidad, lee el resumen y pregúntate: ¿podría explicarle esto a otra persona con mis propias palabras? Si la respuesta es sí, dominas la unidad. Si es no, repasa la explicación detallada antes de seguir.

En cuanto al ritmo, una unidad por día es un paso excelente: en 10 días terminas el libro. Si tienes menos tiempo, una unidad cada dos o tres días también funciona; lo importante es la constancia, no la velocidad. Y cuando completes las 10 unidades, presenta el examen final en serio: sin apuntes, con tiempo, como si fuera un examen real. Si sacas 60 o más, estás listo para el siguiente nivel.

Un consejo importante: no te saltes unidades. Cada una construye sobre la anterior. La unidad 5 de semántica no tiene sentido sin la 3 de contenedores, y el proyecto de la unidad 10 usa todo lo anterior. El orden está diseñado a propósito.

Qué necesitas para empezar

La lista es corta y todo es gratis:

- Una computadora. No tiene que ser potente: cualquier equipo que abra un navegador moderno sirve. Incluso puedes practicar en una tablet, aunque la computadora es más cómoda.

- Un navegador moderno: Chrome, Edge, Firefox o Safari. Todos entienden HTML5 perfectamente.
- Un editor de texto para código. Te recomiendo Visual Studio Code, que es gratis y es el que usan la mayoría de los profesionales. Pero si quieres empezar ya mismo, el Bloc de notas también sirve para tus primeros archivos.
- Conexión a internet para consultar la documentación y tomar el curso en video en lucasegaacademy.com cuando quieras reforzar un tema viendo al profesor explicarlo en el pizarrón.

No necesitas instalar servidores, ni bases de datos, ni nada complicado. HTML5 se abre directo en el navegador: escribes el archivo, le das doble clic, y ves tu página. Esa inmediatez es parte de la magia.

Lo que vas a lograr

Cuando termines este libro y apruebes el examen final, serás capaz de:

- Escribir documentos HTML5 completos y bien estructurados desde cero, sin plantillas.
- Usar correctamente todas las etiquetas fundamentales: texto, enlaces, imágenes, listas, tablas, formularios y multimedia.
- Construir formularios profesionales con validación nativa que funcionan sin JavaScript.
- Escribir HTML semántico que los buscadores entienden y que las personas con discapacidades pueden usar con lectores de pantalla.
- Aplicar buenas prácticas profesionales: código indentado, comentado y fácil de mantener.
- Crear tu propio portafolio web con secciones, blog y formulario de contacto: tu carta de presentación como desarrollador.
- Leer y entender el código HTML de cualquier página web que visites.

Y lo más importante: tendrás la base sólida para seguir con CSS3, que es el siguiente curso de la academia y el que le da estilo y diseño a todo lo que aquí construyas. Después vendrán JavaScript, y la ruta completa del desarrollo frontend. Pero todo, todo empieza aquí, con HTML5.

Unas palabras del autor

Soy Esteban, fundador de Lucas EGA Academy. Creé esta academia con una idea simple: el conocimiento de tecnología debe ser gratis y debe estar en español, explicado de forma clara, sin rodeos y sin suponer que ya sabes cosas que nadie te enseñó.

Este libro es parte de ese compromiso. No te voy a pedir dinero por aprender: el curso en video de 30 clases está gratis en lucasegaacademy.com, y este libro es su compañero escrito. Si este material te sirve y algún día quieres apoyar el proyecto, la academia vende libros en PDF para quienes prefieren el formato digital coleccionable, pero aprender, eso siempre será gratis aquí.

Un último consejo antes de empezar: la programación se aprende programando. Este libro te da el mapa, pero el camino lo recorres tú, tecla por tecla. Va a haber momentos de frustración, eso es normal y le pasa a todos los programadores del mundo, incluidos los que llevan 20 años. La diferencia entre quien aprende y quien se rinde no es la inteligencia: es la constancia. Sigue adelante, practica todos los días aunque sea 30 minutos, y en unas semanas vas a ver tu primera página web publicada y vas a sentir un orgullo que no se compra.

Nos vemos en la Unidad 1. A programar.

UNIDAD 1: Introducción a HTML5 y primeros pasos

Bienvenido al punto donde todo empieza. Antes de que puedas dominar formularios, tablas, multimedia, estilos y scripts, necesitas entender la base sobre la que descansa toda la web moderna: HTML5. Esta primera unidad no es un trámite rápido que debas saltarte; es la cimentación sobre la que vas a levantar todo lo demás. Si la cimentación queda chueca, todo lo que construyas encima se va a tambalear. Si queda firme, cada unidad que sigue se va a sentir natural, casi obvia.

En esta unidad vas a aprender qué es HTML y por qué HTML5 es la versión que manda hoy en día. Vas a conocer la historia del lenguaje desde sus inicios humildes hasta el estándar actual, para que entiendas por qué las cosas son como son y no solo las memorices. Vas a descubrir cómo trabaja un navegador por dentro cuando recibe un documento HTML: cómo lo lee, cómo lo interpreta y cómo lo convierte en la página que ves en pantalla. Vas a estudiar la estructura básica de todo documento web (DOCTYPE, html, head, title y body) hasta que la puedas escribir con los ojos cerrados. Y vas a dominar las etiquetas de texto fundamentales: los encabezados h1 hasta h6, el párrafo p, y las marcas de énfasis strong y em, además de otras útiles como b, i, u, mark, small y br.

Por qué importa tanto esta unidad: HTML no es un tema más del curso, es el tema del curso. Todo lo demás que vas a estudiar en este libro vive dentro de un documento HTML. Las imágenes de la Unidad 4 van dentro del body. Los formularios de la Unidad 6 van dentro del body. Los videos, los audios, las tablas, los scripts: todo se incrusta en el documento HTML que aquí aprendes a construir. Aprender HTML primero es como aprender a sostener el lápiz antes de aprender a dibujar. Sin este paso no hay página posible, así de simple.

Hay otra razón de peso para tomar esta unidad en serio: la web premia a quien domina lo básico. Muchos desarrolladores saben copiar fragmentos de código de internet, pero los que realmente entienden la estructura de un documento, la jerarquía de los encabezados y el significado semántico de cada etiqueta escriben páginas que cargan más rápido, que los buscadores entienden mejor y que las personas con discapacidad visual pueden navegar con lectores de pantalla. Lo básico no es lo aburrido: lo básico es lo poderoso. Google no posiciona bien una página con cinco etiquetas h1 peleándose entre sí. Un lector de pantalla no puede guiar a un usuario ciego si el contenido está armado sin jerarquía. Cuando entiendes el porqué detrás de cada etiqueta, tu código deja de ser adivinanza y se convierte en oficio.

Cómo encaja esta unidad en el curso completo: el "Curso Completo de HTML5" de Lucas EGA Academy está diseñado como una escalera. Esta unidad es el primer peldaño. Aquí aprendes a crear un documento válido, a estructurarlo y a darle formato al texto. En la unidad siguiente subirás al siguiente nivel: enlaces e imágenes, para que tus páginas empiecen a conectarse con el mundo y a mostrar contenido visual. Después vendrán las listas, las tablas, los formularios, el contenido multimedia y los elementos semánticos modernos de HTML5 como header, nav, main, article y footer. Cada unidad nueva usa sin piedad lo que aprendiste en las anteriores. Por eso el plan de estudio no admite huecos: si dominas la estructura básica hoy, mañana solo tendrás que preocuparte por lo nuevo, no por andar reparando la base.

Una nota sobre cómo estudiar esta unidad. No la leas como si fuera una novela: léela con las manos en el teclado. Cada vez que veas un bloque de código, cópialo en tu editor, guárdalo como un archivo con extensión.html y ábrelo en tu navegador. Cambia cosas, rómpelo a propósito, quita una etiqueta de cierre y mira qué pasa. El error es un maestro excelente cuando es barato: aquí romper algo no cuesta nada y te enseña el doble. Los ejercicios de la sección de Práctica están diseñados exactamente para eso, para que falles en pequeño y aprendas en grande. Al final de la unidad encontrarás un resumen con los puntos clave; úsalo para repasar, pero no como sustituto de la práctica.

También quiero que sepas algo desde ahora: no necesitas nada caro ni complicado para seguir este curso. Un editor de texto sencillo y un navegador moderno bastan. En las siguientes secciones te voy a mostrar exactamente cómo preparar tu espacio de trabajo en cinco minutos. La barrera de entrada para crear páginas web es de las más bajas de toda la tecnología, y eso es una noticia buenísima: significa que desde hoy, con lo que ya tienes en tu computadora, puedes empezar a construir cosas reales que se ven en un navegador de verdad.

Así que manos a la obra. Esta unidad es larga a propósito: cada concepto viene explicado a fondo, con analogías de la vida real, con ejemplos de código comentados línea por línea, con errores típicos que cometen los principiantes (y cómo evitarlos) y con buenas prácticas que usan los profesionales. Tómame tu tiempo. Cuando termines, vas a poder crear tu primera página web completa, con estructura correcta, títulos jerarquizados y texto bien formateado. Y eso, créeme, ya es más de lo que mucha gente que dice "saber de páginas web" puede hacer bien. Antes de entrar en materia, déjame contarte exactamente qué vas a encontrar en esta unidad y cómo sacarle el máximo provecho. La unidad está dividida en cinco grandes secciones. La primera es la explicación detallada, donde desarrollamos la teoría con calma: qué es HTML, qué es HTML5, su historia, cómo lo procesa el navegador, la estructura de un documento y las etiquetas de texto. La segunda sección trae ejemplos paso a paso: cuatro

programas completos, comentados línea por línea, para que veas la teoría funcionando. La tercera es la práctica: cinco ejercicios con su solución explicada, para que pongas las manos en el código. Y cerramos con un resumen de los puntos clave para repasar.

Un consejo de estudio que vale oro: no intentes memorizar todas las etiquetas de golpe. Memorizar sin entender es como aprenderse de memoria los nombres de las herramientas sin saber para qué sirve cada una. En cambio, entiende la lógica: HTML es un lenguaje de marcado, lo que significa que envuelves el contenido con etiquetas que dicen qué ES cada cosa, no cómo se ve. Cuando interiorizas esa idea, las etiquetas se vuelven obvias: h1 es "encabezado principal", p es "párrafo", strong es "texto importante". El significado manda, la apariencia viene después con CSS, que es otro curso completo de la academia.

Vamos a preparar tu taller. Necesitas dos herramientas y nada más. Primera: un editor de texto para programadores. Puedes usar el Bloc de notas si estás en Windows, pero te recomiendo algo con resaltado de sintaxis como Visual Studio Code, que es gratis y es el estándar de la industria. El resaltado de sintaxis pinta las etiquetas de un color y el texto de otro, y eso solo ya te ahorra la mitad de los errores de principiante porque ves de un vistazo si olvidaste cerrar algo. Segunda: un navegador moderno como Chrome, Edge o Firefox. Eso es todo. No necesitas servidores, no necesitas instalar nada raro, no necesitas pagar licencias.

El ritual de trabajo que vas a repetir cientos de veces en este curso es este: creas un archivo de texto, le pones un nombre que termine en.html (por ejemplo mi-primera-página.html), escribes tu código, lo guardas y luego lo abres con el navegador haciendo doble clic o arrastrándolo a la ventana del navegador. Cada vez que cambies algo, guardas y recargas la página con la tecla F5. Ese ciclo de escribir, guardar y recargar es el latido del desarrollo web. En cinco minutos lo tienes dominado.

Una advertencia importante sobre dónde escribir el código: nunca uses un procesador de textos como Word para escribir HTML. Word agrega formato invisible (tipos de letra, comillas curvas, estilos) que rompe el código sin que te des cuenta. Usa siempre un editor de texto plano o un editor de código. Este detalle le ha costado horas de frustración a más de un principiante, y ahora tú ya lo sabes desde el día uno.

Hablemos de expectativas realistas. Al terminar esta unidad no vas a construir el próximo Facebook; vas a construir algo mejor para este momento: una página web válida, bien estructurada y con texto correctamente formateado, hecha por ti desde cero. Ese es un logro real. Cada desarrollador profesional que admiras pasó por este exacto punto: un archivo.html, un "Hola Mundo" en pantalla y una sonrisa de oreja a oreja. La diferencia entre ellos y quien se rinde no es el talento, es que ellos siguieron al día siguiente. Tú ya diste el paso más difícil, que es empezar.

Por último, una promesa: todo lo que aprendas aquí lo vas a usar. No hay teoría de adorno en esta unidad. La estructura DOCTYPE-html-head-body aparece en el 100 por ciento de las páginas web del planeta, sin excepción. Los encabezados y párrafos son el pan de cada día de cualquier sitio: noticias, blogs, tiendas, redes sociales, todo. Cuando en unidades futuras agregues imágenes, enlaces y formularios, lo harás dentro de la estructura que hoy vas a dominar. Así que cada minuto que inviertas aquí se multiplica después. Empecemos.

Explicación detallada

Qué es HTML: el lenguaje que describe la web

HTML son las siglas de HyperText Markup Language, que en español significa Lenguaje de Marcado de Hipertexto. Vamos a desarmar ese nombre porque cada palabra te enseña algo. "Hipertexto" es texto que contiene enlaces a otros textos: la idea revolucionaria de que un documento puede saltar a otro con un clic, formando una red. Esa red es la World Wide Web. "Marcado" significa que el lenguaje funciona poniendo marcas, llamadas etiquetas, alrededor del contenido para decir qué es cada parte. Y "lenguaje" porque tiene reglas de sintaxis que debes respetar, aunque no es un lenguaje de programación: no hace cálculos ni toma decisiones, solo describe.

La distinción es importante y conviene dejarla clara desde el principio: HTML no programa, HTML describe. Cuando escribes `<h1>Mi título</h1>`, no le estás diciendo a la computadora "calcula algo" ni "si pasa esto haz aquello". Le estás diciendo: "este texto ES un encabezado de nivel uno, el título principal de la página". HTML es un lenguaje descriptivo y estructural. La lógica y la interactividad vendrán después con JavaScript, y la apariencia visual con CSS, pero la descripción de qué es cada cosa, el esqueleto y los órganos del documento, eso es HTML y nadie más lo hace.

Piensa en HTML como el plano arquitectónico de una casa. El plano no es la casa: no tiene ladrillos ni pintura. Pero dice dónde va cada muro, dónde va cada puerta, qué cuarto es la

cocina y cuál es el baño. Sin el plano, los albañiles no sabrían qué construir. De la misma manera, sin HTML el navegador no sabría qué mostrar: el navegador es el constructor que lee tu plano (el documento HTML) y levanta la página que el usuario ve. Si el plano está bien hecho, la construcción sale bien. Si el plano está incompleto o contradictorio, el constructor improvisa, y ya sabes cómo terminan las construcciones improvisadas.

Otra analogía útil es la de una receta de cocina. Una receta no es el platillo: es una lista de ingredientes con instrucciones ordenadas. HTML es parecido: es una lista de contenidos (textos, imágenes, enlaces) con instrucciones de estructura (esto es un título, esto es un párrafo, esto va dentro de aquello). El navegador es el cocinero que sigue la receta y presenta el platillo final en pantalla. Y como toda receta, si omites un paso o mezclas el orden, el resultado puede salir raro, aunque a veces siga siendo comestible.

Cómo se ve el marcado en la práctica: las etiquetas casi siempre vienen en pares, una de apertura y una de cierre. La de apertura es el nombre entre signos de menor y mayor, como `<p>`, y la de cierre es igual pero con una diagonal antes del nombre, como `</p>`. Todo lo que queda entre las dos es el contenido afectado por esa etiqueta. Mira este ejemplo sencillo:

```
<p>Este texto es un párrafo.</p>
```

La etiqueta `<p>` abre, el texto es el contenido, y `</p>` cierra. El navegador lee eso y entiende: "aquí hay un párrafo", y lo muestra con un espacio antes y después, como los párrafos de un libro. Algunas etiquetas no necesitan cierre porque no envuelven contenido, como `<br>` que es un salto de línea; de esas hablaremos más adelante en esta misma unidad.

Las etiquetas también pueden llevar información extra llamada atributos. Un atributo es un par nombre-valor que va dentro de la etiqueta de apertura y le da detalles al navegador. Por ejemplo, más adelante verás `<html lang="es">`, donde `lang="es"` es un atributo que dice "el idioma de este documento es español". Piensa en los atributos como los adjetivos de la etiqueta: la etiqueta dice qué es, el atributo dice cómo o cuál. En esta unidad los usaremos poco, pero es bueno que el concepto te suene desde ahora porque en las siguientes unidades los atributos serán protagonistas.

Qué es HTML5: la versión moderna y por qué es la que importa

Si HTML es el lenguaje, HTML5 es su quinta y más reciente versión mayor, el estándar actual con el que se construye toda la web moderna. Cuando alguien dice "sé HTML" en 2026, se entiende que sabe HTML5, porque las versiones anteriores quedaron obsoletas hace años. HTML5 no es un lenguaje distinto: es HTML llevado a su forma madura, más potente, más

simple de escribir y con capacidades que antes requerían trucos complicados o programas externos.

Qué trajo de nuevo HTML5 respecto a sus antecesores: primero, simplicidad. La declaración de tipo de documento se redujo a `<!DOCTYPE html>`, una línea corta y memorable, cuando antes era un galimatías largo que nadie se sabía de memoria. Segundo, elementos semánticos nuevos como `header`, `nav`, `main`, `article`, `section` y `footer`, que permiten describir la estructura de la página con significado (esto es el encabezado, esto es la navegación) en lugar de usar divisiones genéricas sin sentido. Tercero, soporte nativo para multimedia: las etiquetas `audio` y `video` permiten incrustar sonido y video sin depender de reproductores externos. Cuarto, nuevos tipos de campos para formularios (`email`, `date`, `number` y más) que le facilitan la vida al usuario en el celular. Y quinto, una especificación pensada para la web de aplicaciones, no solo para documentos estáticos.

Para ti como estudiante, lo más importante de HTML5 es esto: es más fácil de aprender que las versiones viejas y es lo único que vale la pena aprender hoy. Los tutoriales antiguos que enseñan HTML 4 o XHTML usan reglas estrictas y verbosas que ya no aplican. Si sigues este libro, vas por el camino moderno desde el día uno. Todo el curso, las 30 clases y todas las unidades de este libro, están escritas en HTML5.

Un detalle práctico: cuando veas por ahí código viejo con etiquetas como `<font>` o `<center>`, o atributos como `bgcolor`, estás viendo HTML antiguo. En HTML5 esas cosas se hacen con CSS, no con etiquetas. No las uses aunque las veas en tutoriales viejos: son mala práctica moderna y algunos navegadores ya ni las soportan bien. Este libro te enseña el camino actual, no el museo.

Y una aclaración sobre el nombre del curso: "Curso Completo de HTML5" no significa que HTML5 sea difícil o interminable. Significa que lo vamos a cubrir completo, desde este primer "Hola Mundo" hasta técnicas avanzadas, paso a paso, sin saltos. La unidad 1 que estás leyendo corresponde a las primeras clases del curso en video de Lucas EGA Academy, donde todo se explica frente al pizarrón con ejemplos de código. El libro y el curso en video se complementan: aquí tienes la profundidad y la referencia permanente; allá tienes la explicación visual. Ambos son gratis en lucasegaacademy.com.

Historia breve: de HTML a HTML5

Para entender por qué HTML5 es como es, conviene conocer de dónde viene. No es historia de adorno: cada etapa dejó una huella en el lenguaje que todavía ves hoy, y conocerla te evita confundirte cuando encuentres código viejo en internet.

Todo empezó en 1989 con un físico británico llamado Tim Berners-Lee, que trabajaba en el CERN, el laboratorio europeo de física en Suiza. Su problema era muy concreto: los científicos necesitaban compartir documentos e investigaciones entre computadoras de todo el mundo, y no había una forma sencilla de enlazar un documento con otro. Su propuesta fue el hipertexto: documentos conectados por enlaces formando una red, la World Wide Web. Para escribir esos documentos inventó HTML, y para verlos inventó el primer navegador. La primera versión de HTML era rudimentaria: un puñado de etiquetas para títulos, párrafos, listas y enlaces. Pero la idea era tan poderosa que despegó como cohete.

En 1993 apareció Mosaic, el primer navegador popular con interfaz gráfica, y la web explotó. De pronto todo el mundo quería páginas web, y las empresas de navegadores entraron en una carrera salvaje conocida como "la guerra de los navegadores", principalmente entre Netscape y Microsoft Internet Explorer. Cada uno agregaba sus propias etiquetas exclusivas para ganar usuarios, y el resultado fue un caos: una página que se veía bien en un navegador se rompía en otro. Era como si cada marca de televisor necesitara su propio tipo de señal.

Para poner orden, en 1997 se publicó HTML 4, un estándar serio impulsado por el W3C (World Wide Web Consortium), la organización que define los estándares de la web. HTML 4 separó mejor la estructura del diseño y funcionó durante años. Después vino un experimento llamado XHTML alrededor del año 2000, que intentó volver a HTML tan estricto como el lenguaje XML: toda etiqueta debía cerrarse, todo en minúsculas, todo perfecto. En teoría sonaba bien; en la práctica era un dolor de cabeza, porque un solo error de sintaxis podía dejar la página en blanco. Los desarrolladores lo sufrieron y la web no lo adoptó con entusiasmo.

Mientras tanto, la web seguía creciendo y pidiendo más: video, aplicaciones interactivas, páginas que funcionaran en celulares. Un grupo de desarrolladores y empresas, cansados de esperar, formó el WHATWG (Web Hypertext Application Technology Working Group) y empezó a trabajar en lo que sería HTML5, con una filosofía distinta: pragmática, tolerante con errores y pensada para las aplicaciones reales que la gente ya estaba construyendo. Finalmente el W3C se sumó al esfuerzo y en 2014 se publicó la recomendación oficial de HTML5. Desde entonces el estándar vive en evolución continua: ya no hay "HTML6" a la vuelta de la esquina, sino un estándar vivo que se actualiza poco a poco.

Qué te deja esta historia como lección práctica: primera, que HTML5 ganó porque es pragmático, no porque sea perfecto. Los navegadores están diseñados para no romperse ante errores: si olvidas cerrar una etiqueta, el navegador adivina tu intención y muestra algo. Eso es amable, pero es un arma de doble filo, porque te permite escribir código descuidado que "funciona" hasta que un día falla de formas misteriosas. Segunda lección: la web premia la compatibilidad. HTML5 se diseñó para que lo viejo siga funcionando, así que verás código

antiguo por todas partes; tu trabajo es reconocerlo y no imitarlo. Tercera: estás aprendiendo el lenguaje en su mejor momento histórico, maduro, estable y universal. Todo navegador moderno habla HTML5 con fluidez.

Cómo funciona un navegador con el HTML

Ya sabes que HTML describe y que el navegador construye. Ahora veamos con más detalle qué hace el navegador cuando abres una página, porque entender este proceso te convierte en alguien que diagnostica problemas en lugar de alguien que adivina.

El viaje empieza cuando escribes una dirección o haces doble clic en tu archivo.html. El navegador obtiene el documento, que no es más que texto plano lleno de etiquetas. Luego empieza la lectura: el navegador recorre el texto de arriba hacia abajo, como tú lees una carta, y va construyendo en su memoria una representación estructurada del documento. A esa estructura interna se le llama DOM, siglas de Document Object Model, o Modelo de Objetos del Documento. Imagínalo como un árbol genealógico: la etiqueta html es el abuelo, head y body son sus hijos, y dentro de body cuelgan los títulos, párrafos y demás como ramas y hojas. Cada etiqueta se convierte en un "nodo" del árbol, con sus hijos anidados dentro.

Una vez construido el árbol, el navegador calcula cómo se debe ver cada nodo: qué tan grande, de qué color, en qué posición. A eso se le llama renderizado. Con solo HTML, el navegador aplica sus estilos por defecto: los h1 se ven grandes y en negrita, los párrafos con espacio entre ellos, los enlaces en azul y subrayados. Esos estilos por defecto son los que ves cuando abres un HTML sin CSS: no es que la página esté "sin diseño", es que está usando el diseño básico que todo navegador trae de fábrica. Más adelante, CSS te permitirá cambiar cada uno de esos aspectos a tu gusto.

La analogía de la construcción vuelve a servir aquí con más precisión. El documento HTML es el plano. El DOM es la estructura de la obra ya levantada: castillos, muros y losas en su lugar. El renderizado es el acabado: pintura, pisos y ventanas. Y el navegador es a la vez el arquitecto que interpreta el plano, el ingeniero que levanta la estructura y el decorador que le da el acabado, todo en fracciones de segundo, cada vez que abres o recargas una página.

Hay tres ideas de este proceso que te van a servir toda la vida. Primera: el navegador lee de arriba hacia abajo, en orden. Eso significa que el orden de tu código importa: lo que escribas primero aparece primero. Segunda: el navegador es tolerante con errores, pero su tolerancia tiene límites y comportamientos impredecibles. Si escribes `<p>Hola` sin cerrar el párrafo, el navegador lo cerrará por ti donde crea conveniente, y "donde crea conveniente" no siempre es donde tú querías. Tercera: lo que ves en pantalla no es tu código, es la interpretación de tu

código. Por eso dos códigos distintos pueden verse igual, y por eso la herramienta de "inspeccionar elemento" del navegador (clic derecho sobre la página) es tan valiosa: te muestra el DOM real que el navegador construyó, no lo que tú creías haber escrito.

Un experimento que te recomiendo hacer hoy mismo: abre tu primera página en el navegador, haz clic derecho sobre un título y elige "Inspeccionar". Se abrirá un panel mostrando el árbol del documento, con tus etiquetas anidadas como ramas. Ver el DOM con tus propios ojos hace que todo lo que acabas de leer deje de ser teoría y se vuelva algo concreto. Hazlo, en serio: es de esos momentos donde las piezas encajan.

La estructura básica de un documento: el esqueleto de toda página

Toda página web del planeta, desde la más sencilla hasta la más compleja, comparte el mismo esqueleto básico. Apréndetelo de memoria, porque lo vas a escribir miles de veces. Aquí está el documento mínimo válido en HTML5:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi primera página</title>
</head>
<body>
  <h1>Hola Mundo</h1>
  <p>Esta es mi primera página web.</p>
</body>
</html>
```

Vamos a diseccionar cada línea, porque cada una tiene un trabajo específico y ninguna sobra.

La primera línea, `<!DOCTYPE html>`, es la declaración de tipo de documento. No es una etiqueta HTML propiamente dicha: es una instrucción para el navegador que dice "lo que sigue es un documento HTML5, interprételo con las reglas modernas". Es una reliquia histórica simplificada: en versiones antiguas esta línea era un párrafo largo e ilegible con referencias a definiciones formales; HTML5 la redujo a estas 15 letras. Si la omities, el navegador entra en algo llamado "modo peculiar" (quirks mode), donde intenta adivinar cómo mostrar páginas viejas mal hechas, y tu página puede verse ligeramente distinta entre navegadores. Moraleja: escribe siempre `<!DOCTYPE html>` como primera línea, sin excepciones. Cuesta dos segundos y te ahorra dolores de cabeza.

La segunda línea abre la etiqueta `<html>`, que es la raíz del documento: todo, absolutamente todo, va dentro de ella. Es el abuelo del árbol DOM del que hablamos antes. Fíjate en el atributo `lang="es"`: le dice al navegador (y a los buscadores, y a los lectores de pantalla) que el contenido está en español. Esto no cambia lo que se ve, pero sí importa: los lectores de pantalla usan ese dato para pronunciar correctamente, y los buscadores lo usan para mostrar tu página a quien busca en español. Buena práctica desde el día uno: `<html lang="es">`.

Dentro de `html` hay dos grandes secciones: `head` y `body`. El `<head>` es la cabeza del documento: contiene información SOBRE la página, no contenido visible. Es como la ficha técnica de un producto: no es el producto, pero dice qué es, de qué está hecho y cómo manejarlo. Dentro del `head` van el título, la codificación de caracteres, enlaces a hojas de estilo y otros metadatos. Nada de lo que pongas en el `head` se muestra directamente en la página (con la excepción del `title`, que aparece en la pestaña del navegador).

Dentro del `head` de nuestro ejemplo hay dos cosas. Primero, `<meta charset="UTF-8">`: esta etiqueta le dice al navegador que el documento usa la codificación UTF-8, que es la que incluye las tildes, la ñ y los signos de interrogación y exclamación de apertura. Sin esta línea, en algunos navegadores tus tildes podrían verse como símbolos raros. En HTML5 esta etiqueta se escribe así de corta; memorízala junto con el DOCTYPE. Segundo, `<title>Mi primera página</title>`: el título del documento. No aparece dentro de la página, sino en la pestaña del navegador, en los marcadores cuando alguien guarda tu página y en los resultados de Google como el texto azul clicable. El `title` es tu primera herramienta de presentación ante el mundo: hazlo descriptivo y único por página.

Luego viene el `<body>`, el cuerpo del documento: aquí vive todo lo que el usuario ve y con lo que interactúa. Títulos, párrafos, imágenes, enlaces, formularios, videos: todo va dentro del `body`. Si el `head` es la ficha técnica, el `body` es el producto en sí. En nuestro ejemplo el `body` contiene un `<h1>` con "Hola Mundo" y un `<p>` con "Esta es mi primera página web.". Cuando abras este archivo en el navegador verás exactamente eso: un título grande y un párrafo debajo.

La analogía de la casa, aplicada a la estructura: `<!DOCTYPE html>` es el permiso de construcción que dice bajo qué reglamento se edifica. `<html>` es el terreno donde se levanta todo. `<head>` es la oficina del arquitecto con los planos, los permisos y las especificaciones: imprescindible, pero los visitantes no entran ahí. `<title>` es el letrero con el nombre del edificio que se ve desde la calle (la pestaña del navegador). Y `<body>` es el edificio mismo con todos sus cuartos amueblados, lo que la gente recorre y usa.

Errores comunes con la estructura básica, para que los reconozcas cuando los cometes (porque los vas a cometer, todos lo hacemos). Error uno: olvidar el DOCTYPE. La página se ve "casi bien" y no entiendes por qué algo se descuadra en otro navegador. Solución: primera línea siempre. Error dos: poner contenido visible dentro del head, como un <h1> entre <head> y </head>. El navegador, en su tolerancia, lo moverá al body por ti, pero el código queda mal estructurado y confunde a quien lo lea. Regla de oro: si se ve en la página, va en el body. Error tres: olvidar cerrar el html o el body al final. Suele "funcionar", pero es código incompleto y mala costumbre. Error cuatro: escribir el title vacío o dejar el que viene por defecto en alguna plantilla. Tu página aparecerá en las pestañas como "Sin título" y en Google igual: una oportunidad desperdiciada. Error cinco: no declarar el charset y luego sorprenderse de ver las tildes convertidas en jeroglíficos. <meta charset="UTF-8"> siempre, desde hoy.

Buenas prácticas que distinguen al profesional desde el primer archivo: indentación consistente (los cuatro espacios que ves en el ejemplo no los exige el navegador, pero le dicen al lector humano qué está dentro de qué: head y body van indentados dentro de html, y el contenido del body va indentado un nivel más). Nombres de archivo en minúsculas, sin espacios ni tildes (mi-página.html, no "Mi Página.HTML"): los servidores web son quisquillosos con eso. Y un title descriptivo en cada documento, aunque sea tu práctica número uno.

Etiquetas de texto: encabezados h1 a h6

Con la estructura dominada, entramos al contenido: el texto. Las primeras etiquetas de contenido que debes dominar son los encabezados, del <h1> al <h6>. Son seis niveles de títulos que forman la jerarquía de tu página, como los títulos y subtítulos de un libro o de un trabajo escolar.

El <h1> es el título principal de la página: el más grande, el más importante, el que resume de qué trata todo. Cada página debería tener UN solo h1, igual que un libro tiene un solo título en la portada. Luego <h2> marca las secciones principales, <h3> las subsecciones dentro de esas secciones, y así sucesivamente hasta <h6>, el nivel más pequeño y específico. Mira cómo se ve la jerarquía en código:

```
<h1>Curso Completo de HTML5</h1>
<h2>Unidad 1: Introducción</h2>
<h3>Qué es HTML</h3>
<h3>Qué es HTML5</h3>
<h2>Unidad 2: Enlaces e imágenes</h2>
<h3>Enlaces</h3>
<h3>Imágenes</h3>
```

Fíjate en la lógica: el h1 nombra el todo, cada h2 abre un bloque grande, y los h3 detallan dentro de cada bloque. Nunca deberías saltarte niveles por capricho (pasar de h1 a h4 porque "se ve del tamaño que quiero"): la jerarquía describe importancia y estructura, no tamaño visual. Si quieres un texto más pequeño, eso se ajusta con CSS; la etiqueta debe reflejar el nivel real en la estructura.

Por qué importa tanto la jerarquía, más allá de lo visual: los buscadores como Google usan los encabezados para entender de qué trata tu página. Un h1 claro y único les dice el tema principal; los h2 y h3 les muestran cómo está organizado. Una página con la jerarquía bien hecha posiciona mejor que una con títulos puestos al azar. Además, las personas con discapacidad visual navegan con lectores de pantalla que les permiten saltar de encabezado en encabezado, como un índice. Si tus encabezados están en orden, esas personas pueden recorrer tu página con eficiencia; si están desordenados, navegar se vuelve un laberinto. Escribir buenos encabezados no es solo estética: es accesibilidad y es posicionamiento.

La analogía del libro es perfecta aquí: el h1 es el título en la portada, los h2 son los títulos de los capítulos, los h3 los subtítulos dentro de cada capítulo, y así hasta los h6 que serían como los títulos de notas al pie muy específicas. Nadie pondría el título de un capítulo más grande que el título del libro, ni mezclaría capítulos y subcapítulos sin orden. Tu página web merece el mismo respeto por la estructura.

Errores típicos con encabezados: usar varios h1 en la misma página porque "todos mis títulos son importantes". No: si todo es lo más importante, nada lo es. Elige uno. Otro error: usar encabezados solo por su tamaño ("pongo h3 porque se ve bonito de ese tamaño"). El tamaño se controla con CSS; la etiqueta se elige por significado. Y otro clásico: saltarse niveles, como poner h1, luego h4, luego h2. El navegador lo mostrará sin quejarse, pero la estructura lógica queda rota y tanto Google como los lectores de pantalla se confunden.

Etiquetas de texto: párrafos con p

La etiqueta <p> define un párrafo: un bloque de texto sobre una idea. Es la etiqueta que más vas a usar en tu vida como desarrollador web, porque la mayor parte del contenido de internet son párrafos: artículos, descripciones, noticias, instrucciones. El navegador muestra cada párrafo como un bloque separado, con espacio antes y después, igual que en un libro bien maquetado.

```
<p>HTML es el lenguaje con el que se construyen todas las páginas web.</p>
<p>HTML5 es su versión más moderna y potente.</p>
```

Dos reglas importantes sobre los párrafos. Primera: un `<p>` contiene texto y marcado de formato (las etiquetas de énfasis que veremos enseguida), pero no debe contener otros bloques grandes como encabezados o listas. Si necesitas un título seguido de texto, son dos elementos hermanos, no uno dentro del otro. Segunda: el navegador ignora los saltos de línea y los espacios múltiples que escribas dentro del código. Si escribes un párrafo con renglones separados por Enter en tu editor, el navegador los juntará todo en un flujo continuo. Para forzar un salto de línea dentro de un párrafo existe `<br>`, que veremos al final de esta sección. Esta es una de las primeras sorpresas del principiante: en HTML, el formato visual del código no determina el formato visual de la página; lo determinan las etiquetas.

La analogía de la receta vuelve a funcionar: cada `<p>` es un paso o una idea de la receta. No mezclas dos pasos en el mismo renglón apretado; cada idea respira en su propio bloque. Y como en la cocina, el orden de los párrafos cuenta la historia en secuencia: el navegador los muestra de arriba hacia abajo, en el orden en que los escribiste.

Error común: escribir todo el texto de la página suelto dentro del body, sin etiquetas `<p>`, como si el body fuera un documento de Word. El navegador lo muestra, sí, pero como una masa de texto sin estructura: sin espacios entre ideas, sin bloques definidos. Siempre envuelve tus bloques de texto en `<p>`. Otro error: usar varios `<br>` seguidos para separar párrafos en lugar de usar `<p>`. Funciona visualmente, pero es como separar capítulos de un libro con renglones vacíos en vez de ponerles título: se ve parecido y significa algo muy distinto para quien lee el código o usa un lector de pantalla.

Etiquetas de énfasis: **strong** y **em**

Dentro de un párrafo a veces necesitas destacar palabras: algo es importante, algo lleva énfasis. HTML5 te da dos etiquetas con significado para eso: `<strong>` y `<em>`. Se ven parecidas a las de formato visual, pero su diferencia es de fondo y vale la pena entenderla desde ahora.

La etiqueta `<strong>` marca texto con énfasis fuerte, con importancia. El navegador la muestra en negritas por defecto, pero su significado real es "esto es importante". Los lectores de pantalla la anuncian con un tono que transmite importancia, y los buscadores le dan un poco más de peso. Úsala cuando el contenido sea realmente relevante: una advertencia, un dato clave, el nombre del concepto que estás definiendo.

```
<p>Recuerda: <strong>guarda tu trabajo cada cinco minutos</strong> para no perderlo.</p>
```


La etiqueta `<em>` marca texto con énfasis, con acento en la pronunciación. El navegador la muestra en cursiva por defecto, y su significado es "esta palabra lleva la carga del énfasis al leer". Es la diferencia entre decir "YO no dije eso" y "yo no dije ESO": la palabra enfatizada cambia el sentido. Los lectores de pantalla la leen con entonación enfática.

```
<p>El examen es <em>mañana</em>, no la próxima semana.</p>
```

Puedes anidarlas y combinarlas sin problema: `<strong><em>texto muy importante y enfatizado</em></strong>` se vale, siempre que cierres en orden inverso al que abriste (la última que abres es la primera que cierras), como las muñecas rusas.

Otras etiquetas de formato de texto: **b**, **i**, **u**, **mark**, **small** y **br**

Aquí es donde muchos principiantes se confunden, así que vamos con calma. Existen `<b>` y `<i>`, que se ven igual que `strong` y `em` (negrita y cursiva), pero NO tienen el mismo significado. La etiqueta `<b>` solo dice "pon esto en negrita" sin declarar importancia: sirve para resaltar visualmente palabras clave o nombres sin darles peso semántico. La etiqueta `<i>` solo dice "pon esto en cursiva": se usa para términos técnicos, palabras en otro idioma o pensamientos. La regla práctica: si lo que quieres es comunicar importancia o énfasis real, usa `strong` y `em`; si solo quieres un efecto visual o una convención tipográfica, usa `b` e `i`. En HTML5 moderno, `strong` y `em` son la opción preferida en la mayoría de los casos.

```
<p>La palabra <i>software</i> viene del inglés.</p>
<p>El <b>HTML</b> es la base de la web.</p>
```

La etiqueta `<u>` subraya el texto. Úsala con moderación y criterio: en la web, el subrayado es la señal visual universal de "esto es un enlace". Si subrayas texto que no es enlace, confundes al usuario que intentará hacerle clic. HTML5 la reserva para casos especiales, como marcar errores ortográficos o nombres propios en textos chinos. Mi recomendación: casi nunca la necesitas; si quieres subrayar por diseño, hazlo con CSS y solo cuando no parezca enlace.

La etiqueta `<mark>` resalta texto como con un marcador fluorescente amarillo. Es perfecta para mostrar resultados de búsqueda o para llamar la atención sobre una frase dentro de un texto largo. Tiene significado: "esto es relevante en este contexto".

```
<p>Buscaste "pizarrón": se encontraron 3 resultados donde aparece
<mark>pizarrón</mark>.</p>
```

La etiqueta `<small>` muestra texto en tamaño reducido y se usa para la "letra chica": avisos legales, derechos de autor, aclaraciones al pie. No es solo "texto chiquito": comunica que es información secundaria.

```
<p><small>Precios en pesos mexicanos. Sujeto a disponibilidad.</small></p>
```

Y `<br>` es el salto de línea: una etiqueta vacía (sin cierre) que corta la línea donde la pongas. Es útil para direcciones postales, poemas o firmas donde el salto de línea forma parte del contenido. Pero ojo: no la uses para separar párrafos (para eso está `<p>`) ni para crear espacio vertical (para eso está CSS). Abusar de `<br><br><br>` para "bajar" contenido es uno de los vicios más comunes del principiante y uno de los primeros que debes erradicar.

```
<p>Lucas EGA Academy<br>Av. de la Web 123<br>Ciudad de México</p>
```

Un apunte sobre etiquetas vacías: `<br>` pertenece a la familia de etiquetas que no envuelven contenido y por tanto no llevan etiqueta de cierre. En HTML5 se escriben simplemente `<br>`, sin la diagonal final que exigía XHTML (`<br />`). Si ves `<br />` en código viejo, es lo mismo; la forma moderna es `<br>` a secas.

Errores comunes de esta sección: usar `<b>` para todo lo importante en vez de `<strong>` (pierdes el significado semántico y la accesibilidad); usar `<i>` donde corresponde `<em>`; encadenar `<br>` para simular párrafos; y subrayar con `<u>` textos que no son enlaces. La buena práctica es preguntarte siempre: ¿quiero decir que esto es importante (strong), que lleva énfasis al leer (em), o solo cambiar su apariencia (b, i)? La respuesta te da la etiqueta correcta.

El anidamiento: las etiquetas como muñecas rusas

Hay una regla de oro en HTML que gobierna cómo se combinan las etiquetas: el anidamiento correcto. Las etiquetas pueden vivir dentro de otras etiquetas, formando capas, pero deben cerrarse en orden inverso al que se abrieron. Es exactamente como las muñecas rusas (matrioskas): la más pequeña se guarda dentro de la mediana, y la mediana dentro de la grande; para cerrarlas, primero cierras la pequeña.

```
<p>Este párrafo tiene <strong>una parte importante</strong> y sigue.</p>
```

Aquí `<strong>` se abrió dentro de `<p>` y se cerró antes de cerrar `<p>`. Perfecto. Lo incorrecto sería cruzarlas:

```
<p>Este párrafo tiene <strong>una parte importante</p></strong>
```

Esto es anidamiento cruzado y está mal. El navegador, en su infinita tolerancia, intentará arreglarlo por ti y probablemente mostrará algo parecido a lo que querías. Pero el DOM resultante no es el que escribiste, y en documentos grandes estos cruces producen comportamientos extraños: estilos que se aplican donde no deben, scripts que no encuentran los elementos, validaciones que fallan. La regla es simple y no admite excepciones: la última etiqueta que abres es la primera que cierras.

Piensa también en el anidamiento como cajas dentro de cajas en una mudanza. Puedes meter una caja pequeña dentro de una grande, y la grande dentro del camión. Lo que no puedes hacer es que la caja grande quede mitad dentro y mitad fuera de la pequeña: físicamente no tiene sentido. En HTML tampoco.

Hay además reglas sobre qué puede ir dentro de qué. Ya vimos que un `<p>` no debería contener encabezados ni otros `<p>`. En cambio, un `<body>` puede contener casi todo. Estas reglas se llaman "modelo de contenido" y las irás aprendiendo etiqueta por etiqueta a lo largo del curso. Por ahora quédate con dos: respeta el orden de cierre y no metas bloques grandes dentro de un párrafo.

Comentarios en HTML: notas para humanos

Los comentarios son notas que escribes en el código para ti o para otros desarrolladores, y que el navegador ignora por completo: no se muestran en la página. Se escriben entre `<!--` y `-->`:

```
<!-- Este es un comentario: el navegador no lo muestra -->
<p>Este párrafo sí se muestra.</p>
```

Los comentarios son una herramienta profesional, no un adorno. Úsalos para marcar secciones ("`<!-- Encabezado principal -->`"), para explicar por qué hiciste algo raro ("`<!-- br necesario para el formato del poema -->`"), o para desactivar temporalmente código mientras pruebas (envuelves el bloque en un comentario y el navegador lo salta). Pero no abuses: un código bien escrito con etiquetas semánticas claras necesita pocos comentarios. Comentar cada línea es tan malo como no comentar nada, porque el ruido esconde lo importante.

Un detalle curioso: como los comentarios viajan dentro del archivo.html, cualquier visitante puede verlos con "Ver código fuente" del navegador. Nunca escribas en un comentario contraseñas, datos personales o información confidencial. Los comentarios son para notas de desarrollo, no para secretos.

Errores comunes recopilados: la galería de los tropiezos

Después de años enseñando, estos son los errores que veo una y otra vez en los principiantes. Te los presento juntos para que los reconozcas al instante cuando aparezcan en tu propio código, porque van a aparecer: es parte del aprendizaje.

Primero, olvidar cerrar etiquetas. Escribes `<h1>Título` y te sigues con el resto. El navegador cierra el h1 donde le parece, y de pronto todo tu texto se ve gigante y en negrita. Diagnóstico: si algo se ve con el formato equivocado "desde cierto punto en adelante", casi seguro olvidaste un cierre justo antes de ese punto. La solución es disciplinarte: cada vez que abras una etiqueta, escribe inmediatamente su cierre y luego escribe el contenido en medio.

Segundo, confundir `<` con `>` o escribir mal el nombre de la etiqueta, como `<h1>Título</h2>` o `<parrafo>`. El navegador no reconoce `<parrafo>` y lo trata como un elemento desconocido sin formato, mostrándolo como texto plano. Si algo "no agarra formato", revisa letra por letra el nombre de la etiqueta.

Tercero, el anidamiento cruzado que ya vimos: `<p><strong>texto</p></strong>`. El navegador lo "arregla" a su manera y el resultado te desconcierta. Cierra siempre en orden inverso.

Cuarto, poner contenido visible en el head. Escribes un `<h1>` entre `<head>` y `</head>` porque "ahí va lo primero". El navegador lo rescata y lo muestra, pero tu código miente sobre su estructura. Todo lo visible va en el body, sin excepción.

Quinto, usar etiquetas por su apariencia en vez de por su significado: h3 porque "se ve del tamaño que quiero", b en vez de strong, br en vez de p. A corto plazo se ve igual; a largo plazo tu código es frágil, inaccesible y difícil de mantener. Elige siempre por significado.

Sexto, no declarar el charset y ver las tildes convertidas en símbolos extraños. `<meta charset="UTF-8">` en el head de cada documento, siempre.

Séptimo, guardar el archivo con la extensión equivocada (página.txt en vez de página.html) o abrir el archivo equivocado en el navegador y preguntarse por qué los cambios "no se reflejan". Revisa siempre qué archivo estás editando y cuál estás viendo.

Octavo, editar en Word o en otro procesador con formato enriquecido, que convierte tus comillas rectas en curvas y agrega caracteres invisibles. Usa un editor de código, punto.

Cada uno de estos errores tiene algo en común: el navegador intenta ayudarte ocultándolo, y esa ayuda te roba la oportunidad de aprender. Por eso te recomiendo validar tu código periódicamente con el validador del W3C (validator.w3.org): le pegas tu código o la dirección de tu página y te dice exactamente qué está mal, con explicaciones. Es como un maestro exigente pero justo, disponible gratis las 24 horas.

Buenas prácticas del profesional desde el día uno

Cerrar con buenas prácticas porque los hábitos se forman al principio y después cuestan cambiar. Estas son las que separan al aficionado del profesional, y ninguna requiere saber más de lo que ya viste en esta unidad.

Escribe siempre en minúsculas los nombres de etiquetas y atributos: `<p>`, no `<P>`. HTML5 lo permite en mayúsculas, pero todo el mundo escribe en minúsculas y tu código debe parecerse al de todo el mundo para que otros lo lean sin fricción.

Cierra todas las etiquetas que abras, aunque el navegador te perdone si no lo haces. El código completo es código predecible.

Indentar con consistencia: cada nivel de anidamiento un poco más adentro. No es obligatorio para el navegador, pero es obligatorio para los humanos que lean tu código, empezando por ti mismo dentro de tres meses cuando ya no recuerdes qué escribiste.

Un solo `h1` por página, jerarquía de encabezados sin saltos, párrafos para bloques de texto, `strong` y `em` para énfasis con significado. Ya conoces las razones: posicionamiento, accesibilidad y claridad.

El `title` siempre descriptivo y único. Piensa que es lo primero que ve Google y lo primero que ve el usuario en su pestaña.

El atributo `lang` en la etiqueta `html` y el meta `charset` en el `head`, en cada documento, sin falta. Son dos líneas que mejoran accesibilidad, posicionamiento y correcta visualización de caracteres.

Nombres de archivo sensatos: minúsculas, sin espacios, sin tildes ni caracteres raros, con extensión `.html`. Usa guiones para separar palabras: `mi-primer-página.html`.

Y la más importante de todas: prueba en el navegador temprano y seguido. No escribas cincuenta líneas y luego pruebes; escribe cinco, guarda, recarga, verifica. Los errores

pequeños se encuentran en segundos; los errores acumulados se esconden entre ellos. El ciclo escribir-guardar-recargar es tu mejor amigo: úsalo sin moderación.

Ejemplos paso a paso

La teoría sin práctica es como aprender a nadar leyendo un manual: entiendes los movimientos pero te hundes en la alberca. Estos cuatro ejemplos están diseñados para que los escribas tú mismo, línea por línea, en tu editor. Cada uno viene comentado para que sepas exactamente qué hace cada parte. No los copies y pegues a ciegas: escríbelos, guárdalos como archivos.html y ábrelos en tu navegador. Después juega con ellos: cambia textos, agrega etiquetas, rompe cosas a propósito y observa qué pasa.

Ejemplo 1: el documento mínimo válido

Este es el "Hola Mundo" de la web: el documento más pequeño que sigue siendo correcto y completo en HTML5. Si entiendes cada línea de este ejemplo, entiendes la estructura que comparten todas las páginas del planeta.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mi primera página web</title>
</head>
<body>
  <h1>Hola Mundo</h1>
  <p>Esta es mi primera página web hecha con HTML5.</p>
</body>
</html>
```

Explicación línea por línea. La línea 1, `<!DOCTYPE html>`, declara que esto es un documento HTML5 para que el navegador use las reglas modernas. La línea 2 abre `<html lang="es">`, la raíz del documento, indicando que el contenido está en español. La línea 3 abre el `<head>`, la sección de información sobre la página. La línea 4, `<meta charset="UTF-8">`, garantiza que las tildes y la ñ se vean bien. La línea 5 define el `<title>`, que aparecerá en la pestaña del navegador como "Mi primera página web". La línea 6 cierra el head. La línea 7 abre el `<body>`, donde vive todo lo visible. La línea 8 muestra un `<h1>` con el título principal "Hola Mundo". La línea 9 muestra un párrafo `<p>` con texto descriptivo. Las líneas 10 y 11 cierran el body y el html, en orden inverso al de apertura, como mandan las muñecas rusas.

Qué vas a ver en el navegador: una pestaña titulada "Mi primera página web", un encabezado grande que dice "Hola Mundo" y debajo un párrafo de texto normal. Parece poco, pero es un documento válido al 100 por ciento: tiene declaración de tipo, idioma, codificación, título y contenido estructurado. Todo lo que construyas de aquí en adelante será este mismo esqueleto con más contenido dentro del body.

Experimento sugerido: quita la línea del DOCTYPE, guarda y recarga. La página se verá igual en la mayoría de los casos, y esa es exactamente la trampa: el problema no se ve hoy, se ve después, cuando algo se descuadre entre navegadores. Vuelve a ponerla. Luego quita el cierre `</p>` y observa que el navegador lo tolera. Vuelve a ponerlo. Estás entrenando el ojo para distinguir entre "funciona" y "está bien hecho".

Ejemplo 2: página con jerarquía de títulos

Este ejemplo muestra cómo organizar el contenido de una página usando los seis niveles de encabezado junto con párrafos. Es el esqueleto típico de un artículo, una entrada de blog o una página informativa.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Guía de la ciudad - Visita el centro</title>
</head>
<body>
  <h1>Guía turística del centro de la ciudad</h1>
  <p>Bienvenido a la guía más completa del centro histórico: qué ver, dónde comer y cómo moverte.</p>

  <h2>Lugares que no te puedes perder</h2>
  <p>El centro concentra la historia de la ciudad en unas cuantas cuadras. Estos son los imprescindibles.</p>

  <h3>La catedral</h3>
  <p>Construida a lo largo de tres siglos, su fachada mezcla estilos que cuentan la historia del país.</p>

  <h3>El palacio municipal</h3>
  <p>Su balcón principal es el escenario del grito de independencia cada 15 de septiembre.</p>

  <h2>Dónde comer</h2>
  <p>La oferta va desde fondas tradicionales hasta restaurantes de autor. Aquí una selección por antojo.</p>

  <h3>Antojitos</h3>
  <p>Los tacos al pastor de la esquina del mercado llevan cuarenta años con la misma receta.</p>

  <h3>Restaurantes</h3>
  <p>Para una cena especial, reserva en la terraza con vista a la catedral.</p>

  <h2>Cómo moverte</h2>
  <p>El centro se recorre mejor a pie. Usa el metro para llegar y camina una vez adentro.</p>
</body>
</html>

```

Observa la arquitectura de la información. Hay un solo h1 que nombra toda la página: la guía turística. Cada gran tema (lugares, comida, transporte) es un h2. Cada punto específico dentro de un tema es un h3. Los párrafos acompañan a cada encabezado con el detalle. Si leyeras

solo los encabezados en orden, entenderías el índice completo de la página sin leer ni un párrafo: h1 guía, h2 lugares, h3 catedral, h3 palacio, h2 comida, h3 antojitos, h3 restaurantes, h2 transporte. Eso es una jerarquía bien construida: funciona como tabla de contenido.

Nota también las líneas en blanco entre bloques en el código. El navegador las ignora, pero al lector humano le ayudan a ver dónde termina una sección y empieza otra. Es indentación y espaciado al servicio de la legibilidad, una buena práctica que no cuesta nada.

Experimento sugerido: cambia uno de los h3 por un h1 y recarga. Visualmente tendrás dos títulos gigantes peleando por atención, y la estructura lógica se rompe: ahora parece que la página trata de dos temas principales. Devuélvelo a h3. Luego prueba saltarte un nivel (h2 seguido directo de h4) y piensa: ¿la página se ve mal? No. ¿La estructura tiene sentido? Tampoco. Esa es la lección: el navegador no te regaña, por eso tú debes ser tu propio inspector de calidad.

Ejemplo 3: artículo con énfasis y formato de texto

Aquí practicamos strong, em, b, i, mark, small y br dentro de un artículo realista. Fíjate en que cada etiqueta de formato se eligió por su significado, no por cómo se ve.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>El café de la mañana - Artículo</title>
</head>
<body>
  <h1>Por qué el café de la mañana sabe mejor</h1>
  <p>Hay una explicación científica detrás de ese primer sorbo: después de horas de ayuno, tus papilas están <em>especialmente</em> sensibles a los sabores.</p>

  <h2>La química del primer sorbo</h2>
  <p>El café contiene más de <b>800 compuestos aromáticos</b>, y el cerebro en ayunas los registra con mayor intensidad. <strong>No es imaginación tuya:</strong> la primera taza objetivamente sabe más intensa que la tercera.</p>

  <h2>El ritual importa</h2>
  <p>La palabra <i>ritual</i> no es casualidad: preparar el café con calma reduce el estrés antes de empezar el día. Los baristas lo saben y por eso el <mark>método de preparación</mark> cambia tanto el resultado final.</p>

  <h2>Una nota responsable</h2>
  <p>Disfruta tu café, pero con medida: más de cuatro tazas al día puede afectar tu sueño.</p>
  <p><small>Este artículo es informativo y no sustituye el consejo de un profesional de la salud.</small></p>

  <h2>Contacto del autor</h2>
  <p>María González<br>maría@ejemplo.com<br>Ciudad de México</p>
</body>
</html>
```

Analicemos las decisiones de formato una por una. En el primer párrafo, `<em>especialmente</em>` lleva el énfasis de la frase: si lo lees en voz alta, esa es la palabra que acentúas. En el segundo párrafo, `<b>800 compuestos aromáticos</b>` resalta un dato llamativo sin declararlo "importante" en sentido semántico: es un gancho visual. En cambio, `<strong>No es imaginación tuya:</strong>` sí declara importancia: es la idea clave del párrafo, lo que el lector debe llevarse. En el tercer párrafo, `<i>ritual</i>` marca un término que se está usando en sentido especial, casi técnico. Y `<mark>método de preparación</mark>` resalta con marcador amarillo el concepto sobre el que el lector debería poner atención, como si alguien lo hubiera subrayado con fluorescente. El `<small>` del aviso legal comunica "letra chica": información secundaria pero necesaria. Y los `<br>` del bloque de contacto separan líneas donde el salto forma parte del contenido (una dirección), no para crear párrafos falsos.

Este ejemplo también muestra anidamiento correcto en acción: cada etiqueta de formato se abre y se cierra dentro de su párrafo, sin cruces. Y muestra que un mismo párrafo puede mezclar varias etiquetas de formato sin problema, siempre que cada una abra y cierre en orden.

Experimento sugerido: cambia el `<strong>` por `<b>` y el `<em>` por `<i>`. La página se verá idéntica, y esa es la trampa visual: por fuera nada cambia, pero por dentro perdiste el significado. Un lector de pantalla ya no anunciará la importancia ni el énfasis. Recuerda: en HTML eliges por significado, no por apariencia.

Ejemplo 4: página personal sencilla

Cerremos los ejemplos con una página completa y realista: una presentación personal. Integra todo lo aprendido: estructura, jerarquía de encabezados, párrafos y formato. Esta podría ser tu primera página "de verdad".

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Sofía Ramírez - Diseñadora web</title>
</head>
<body>
  <!-- Presentación principal -->
  <h1>Sofía Ramírez</h1>
  <p>Hola, soy <strong>Sofía Ramírez</strong>, diseñadora web de Guadalajara.
  Convierto ideas en páginas rápidas, claras y bonitas.</p>

  <!-- Servicios -->
  <h2>Lo que hago</h2>

  <h3>Diseño de páginas</h3>
  <p>Diseño sitios para negocios locales: <em>rápidos</em>, adaptados al celular
  y fáciles de actualizar.</p>

  <h3>Rediseño</h3>
  <p>Tomo tu página actual y la modernizo sin que pierdas tu contenido ni tu
  posicionamiento. El <mark>primer diagnóstico es gratis</mark>.</p>

  <!-- Trayectoria -->
  <h2>Mi historia</h2>
  <p>Empecé aprendiendo <b>HTML</b> por curiosidad en 2024. Lo que era un
  pasatiempo se convirtió en mi profesión cuando mi primer cliente me recomendó con
  tres más.</p>
  <p>Hoy he trabajado con <strong>más de 20 negocios</strong> y sigo aprendiendo
  todos los días, porque la web no se detiene.</p>

  <!-- Contacto -->
  <h2>Contacto</h2>
  <p>Escríbeme y platiquemos de tu proyecto:<br>sofía@ejemplo.com<br>Guadalajara,
  Jalisco</p>
  <p><small>Página personal de demostración creada con HTML5.</small></p>
</body>
</html>

```

Qué hace bien esta página: el h1 es el nombre de la persona, el tema único de la página. Los h2 dividen en tres bloques lógicos: servicios, historia y contacto. Los h3 detallan dentro de servicios. Cada bloque combina párrafos descriptivos con formato significativo: strong para el nombre y el logro ("más de 20 negocios"), em para el énfasis en "rápidos", mark para destacar

la oferta del diagnóstico gratis, b para el término HTML como palabra clave visual, small para la nota al pie y br para los datos de contacto. Los comentarios `<!-- comentario -->` marcan las secciones para quien lea el código, sin afectar lo que se ve.

Esta página, tal cual, ya es publicable: es válida, está estructurada, tiene título descriptivo, idioma declarado y codificación correcta. Con lo que sabes al terminar esta unidad puedes hacerla tuya: cambia el nombre, los textos y los datos, y tendrás tu primera página personal real. Más adelante, con CSS, la harás bonita; con enlaces e imágenes, la harás rica. Pero la base, la que sostiene todo, ya la tienes aquí.

Práctica

Llegó la hora de ensuciarte las manos. Estos cinco ejercicios están ordenados de menor a mayor dificultad y cubren todo lo visto en la unidad. Hazlos en orden, sin mirar la solución hasta haberlo intentado de verdad: la solución está para verificar y para aprender, no para copiar. Para cada ejercicio crea un archivo.html nuevo, escribe tu código, guárdalo y ábrelo en el navegador.

Ejercicio 1: tu documento mínimo

Enunciado: crea un archivo llamado ejercicio1.html que sea un documento HTML5 válido y mínimo. Debe incluir la declaración DOCTYPE, la etiqueta html con idioma español, un head con codificación UTF-8 y un title que diga "Ejercicio 1 - Mi inicio", y un body con un encabezado h1 que diga tu nombre y un párrafo que diga en una frase qué esperas aprender en este curso.

Solución explicada paso a paso. Paso 1: abre tu editor y crea un archivo nuevo; guárdalo de inmediato como ejercicio1.html para no olvidar la extensión. Paso 2: escribe la primera línea, `<!DOCTYPE html>`, la declaración que pone al navegador en modo moderno. Paso 3: abre `<html lang="es">` en la línea 2, declarando el español como idioma. Paso 4: abre el `<head>` y dentro coloca `<meta charset="UTF-8">` para las tildes y `<title>Ejercicio 1 - Mi inicio</title>` para la pestaña; luego cierra el head. Paso 5: abre el `<body>` y dentro escribe `<h1>Tu Nombre</h1>` (con tu nombre real) y `<p>Quiero aprender a crear páginas web desde cero.</p>` (con tu propia frase). Paso 6: cierra el body y luego el html, en ese orden. Paso 7: guarda y abre el archivo en el navegador; verifica que la pestaña diga "Ejercicio 1 - Mi inicio", que tu nombre se vea grande y que el párrafo aparezca debajo. El código completo queda así:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Ejercicio 1 - Mi inicio</title>
</head>
<body>
  <h1>Ana Torres</h1>
  <p>Quiero aprender a crear páginas web desde cero y publicar mi portafolio.</p>
</body>
</html>
```

Si algo no se ve como esperas, revisa la lista de errores comunes: etiquetas sin cerrar, contenido en el head, archivo guardado con otra extensión. Nueve de cada diez problemas del principiante están en esa lista.

Ejercicio 2: jerarquía de una receta

Enunciado: crea receta.html con la receta de tu platillo favorito usando una jerarquía correcta de encabezados. Requisitos: un solo h1 con el nombre del platillo; al menos dos h2 para las grandes secciones (por ejemplo "Ingredientes" y "Preparación"); al menos dos h3 repartidos dentro de esas secciones (por ejemplo "Para la salsa" dentro de ingredientes, o pasos numerados por grupos dentro de preparación); y párrafos descriptivos donde tenga sentido. El title debe describir la página.

Solución explicada paso a paso. Paso 1: decide la estructura en papel antes de codificar: h1 el platillo, h2 ingredientes y preparación, h3 los subgrupos. Planear la jerarquía antes de escribir es un hábito profesional. Paso 2: crea el esqueleto del documento (DOCTYPE, html lang es, head con charset y title, body) como en el ejercicio 1. Paso 3: escribe el h1 con el nombre del platillo. Paso 4: escribe el h2 "Ingredientes", luego un h3 "Para la masa" seguido de un párrafo con esos ingredientes, y un h3 "Para la salsa" con su párrafo. Paso 5: escribe el h2 "Preparación" con párrafos que describan los pasos en orden. Paso 6: verifica que no haya saltos de nivel (nunca h1 directo a h3) y que haya un solo h1. Paso 7: abre en el navegador y lee solo los encabezados en orden: si forman un índice coherente, la jerarquía está bien.

Ejemplo de resultado:

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Receta de enchiladas verdes</title>
</head>
<body>
  <h1>Enchiladas verdes</h1>
  <p>La receta de la abuela, con el toque de chile serrano que las hace inolvidables.</p>

  <h2>Ingredientes</h2>

  <h3>Para las tortillas y el relleno</h3>
  <p>12 tortillas de maíz, 500 gramos de pollo deshebrado, crema y queso fresco al gusto.</p>

  <h3>Para la salsa verde</h3>
  <p>8 tomates verdes, 2 chiles serranos, un diente de ajo, un trozo de cebolla y sal.</p>

  <h2>Preparación</h2>
  <p>Cuece los tomates y los chiles, licúalos con el ajo y la cebolla, y fríe la salsa lightly con un poco de aceite.</p>
  <p>Rellena las tortillas con el pollo, báñalas con la salsa caliente y corona con crema y queso.</p>
</body>
</html>

```

Nota: la palabra "lightly" se me coló en inglés; en tu versión escribe todo en español ("fríe la salsa ligeramente"). Los detalles del idioma también cuentan en una página profesional.

Ejercicio 3: artículo con énfasis significativo

Enunciado: crea artículo.html con un mini artículo de tres párrafos sobre un tema que te apasione. Requisitos: h1 con el título del artículo; usa al menos dos veces para ideas realmente importantes; usa al menos dos veces para palabras que llevarían énfasis al leer en voz alta; usa <mark> una vez para resaltar un dato o frase clave; e incluye un párrafo final con <small> para una nota al pie. Cada etiqueta debe estar justificada por su significado, no por su apariencia.

Solución explicada paso a paso. Paso 1: elige tu tema y escribe primero el texto plano de los tres párrafos, sin formato. Paso 2: relee y marca con lápiz (mental o real) qué ideas son las más importantes del artículo: esas van con strong. Paso 3: identifica qué palabras acentuarías si lo leyeras en voz alta: esas van con em. Paso 4: elige el dato o frase que un lector apurado debería captar de un vistazo: ese va con mark. Paso 5: redacta la nota al pie (fuente, aclaración o descargo) con small. Paso 6: arma el documento completo con su estructura base y verifica el anidamiento: cada etiqueta de formato abre y cierra dentro de su párrafo. Ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Por qué correr cambió mis mañanas</title>
</head>
<body>
  <h1>Por qué correr cambió mis mañanas</h1>
  <p>Empecé a correr por salud y me quedé por la <em>claridad mental</em>: treinta minutos de trote me ordenan las ideas mejor que cualquier lista de pendientes.</p>
  <p><strong>La constancia importa más que la velocidad.</strong> Correr lento cinco días seguidos transforma más que un sprint heroico una vez al mes. Mi marca personal mejoró <mark>12 minutos en 10 kilómetros</mark> en seis meses, sin entrenamientos extremos.</p>
  <p>Si vas a empezar, hazlo gradual: <strong>escucha a tu cuerpo</strong> y aumenta la distancia <em>poco a poco</em>, semana a semana.</p>
  <p><small>Experiencia personal; consulta a un profesional antes de iniciar actividad física intensa.</small></p>
</body>
</html>
```

Fíjate en las decisiones: strong para las dos ideas capitales (la constancia y escuchar al cuerpo), em para las palabras con carga al leer en voz alta (claridad mental, poco a poco), mark para el dato que brilla (12 minutos en 10 kilómetros) y small para la nota responsable. Si puedes explicar por qué cada etiqueta es la correcta, dominaste la lección.

Ejercicio 4: detecta y corrige los errores

Enunciado: el siguiente código tiene cinco errores de los que vimos en la unidad (etiquetas sin cerrar, anidamiento cruzado, contenido en el head, jerarquía rota y falta de charset). Cópialo a errores.html, encuentra los cinco errores, corrígelos y verifica que la página se vea bien.


```

<!DOCTYPE html>
<html>
<head>
  <title>Página con errores</title>
  <h1>Este título está en el lugar equivocado</h1>
</head>
<body>
  <h1>Mi página</h1>
  <h4>Sección principal</h4>
  <p>Este párrafo tiene <strong>un error de anidamiento</strong></p>
  <p>Este párrafo nunca se cierra
</body>
</html>

```

Solución explicada paso a paso. Paso 1: copia el código tal cual y ábrelo en el navegador para ver cómo el navegador "tolera" los errores: notarás cosas raras que te darán pistas. Paso 2: error uno, falta el atributo de idioma y el charset. Agrega lang="es" a la etiqueta html y <meta charset="UTF-8"> dentro del head. Paso 3: error dos, hay un h1 dentro del head ("Este título está en el lugar equivocado"). Muévelo al body o elimínalo; además había dos h1, lo cual rompe la regla de un solo título principal. Deja un único h1 en el body. Paso 4: error tres, el h4 cuelga directo del h1 saltándose niveles. Cámbialo a h2 para mantener la jerarquía continua. Paso 5: error cuatro, anidamiento cruzado en <p>......</p>. Corrige el orden de cierre: <p>Este párrafo tiene un error de anidamiento</p>. Paso 6: error cinco, el último párrafo no tiene </p>. Agrégalo. El documento corregido queda así:

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Página con errores corregidos</title>
</head>
<body>
  <h1>Mi página</h1>
  <h2>Sección principal</h2>
  <p>Este párrafo tiene <strong>un error de anidamiento</strong></p>
  <p>Este párrafo ahora sí se cierra</p>
</body>
</html>

```

Paso 7: válida el resultado en validator.w3.org para confirmar que no quedan errores. Acostúmbrate a validar: es el hábito que separa al que cree que su código está bien del que lo sabe.

Ejercicio 5: tu página de presentación completa

Enunciado: integra todo lo aprendido en una página personal real llamada `presentación.html`.

Requisitos: estructura completa y válida (DOCTYPE, html lang es, head con charset y title descriptivo); un h1 con tu nombre; un h2 "Sobre mí" con dos párrafos (usa strong y em con criterio); un h2 "Mis metas" con un h3 "Este mes" y un h3 "Este año", cada uno con su párrafo; un h2 "Contacto" con tus datos usando br para los saltos de línea; y una nota final con small. Además, incluye dos comentarios HTML que marquen secciones del código.

Solución explicada paso a paso. Paso 1: planea en papel la jerarquía: h1 tu nombre; h2 Sobre mí, Mis metas, Contacto; h3 Este mes y Este año dentro de Mis metas. Paso 2: escribe el esqueleto base del documento. Paso 3: desarrolla cada sección con su contenido real: escribe sobre ti de verdad, no texto de relleno; las páginas reales se sienten distintas y te motivan más. Paso 4: aplica el formato con significado: strong para lo importante de tu historia, em para el énfasis natural, mark si quieres destacar un dato (opcional), small para la nota final, br solo en el bloque de contacto. Paso 5: agrega los dos comentarios marcando secciones, por ejemplo `<!-- Sobre mí -->` antes del h2 correspondiente. Paso 6: revisa la lista de verificación final: un solo h1, sin saltos de jerarquía, todas las etiquetas cerradas, anidamiento en orden, title descriptivo, lang y charset presentes, archivo en minúsculas con extensión.html. Paso 7: ábrela en el navegador, válidala en el W3C y presúmela: es tu primera página web completa y correcta. Ejemplo de referencia (adáptalo con tus datos):

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Diego Luna - Mi presentación</title>
</head>
<body>
  <!-- Encabezado y presentación -->
  <h1>Diego Luna</h1>

  <h2>Sobre mí</h2>
  <p>Soy <strong>Diego Luna</strong>, estudiante de desarrollo web en la
  <em>hermosa</em> ciudad de Monterrey. Descubrí el HTML hace un mes y ya no puedo
  parar.</p>
  <p>Mi objetivo es <strong>claro</strong>: dominar la base para después
  construir aplicaciones completas.</p>

  <!-- Metas -->
  <h2>Mis metas</h2>

  <h3>Este mes</h3>
  <p>Terminar el <mark>Curso Completo de HTML5</mark> de Lucas EGA Academy y
  publicar mis primeras tres páginas.</p>

  <h3>Este año</h3>
  <p>Aprender CSS y JavaScript para convertirme en desarrollador
  <em>completo</em> y conseguir mi primer cliente.</p>

  <!-- Contacto -->
  <h2>Contacto</h2>
  <p>Puedes escribirme a:<br>diego@ejemplo.com<br>Monterrey, Nuevo León</p>
  <p><small>Página creada como práctica de la Unidad 1 del Curso Completo de
  HTML5.</small></p>
</body>
</html>

```

Si completaste los cinco ejercicios, ya hiciste más práctica real que la mayoría de los tutoriales de internet: escribiste documentos válidos, jerarquizaste contenido, aplicaste formato con significado, depuraste errores ajenos e integraste todo en una página propia. Eso es exactamente lo que hace un desarrollador web todos los días, solo que en proyectos más grandes.

Resumen de la unidad

- HTML (HyperText Markup Language) es el lenguaje de marcado con el que se construyen todas las páginas web: describe qué ES cada parte del contenido mediante etiquetas, no es un lenguaje de programación.
- HTML5 es la versión moderna y actual del lenguaje: más simple de escribir, con elementos semánticos, multimedia nativa y formularios mejorados; es lo único que vale la pena aprender hoy.
- El navegador lee el documento de arriba hacia abajo, construye el árbol DOM con las etiquetas anidadas y luego renderiza la página aplicando sus estilos por defecto; lo que ves en pantalla es la interpretación de tu código.
- Todo documento HTML5 válido empieza con `<!DOCTYPE html>`, seguido de `<html lang="es">` como raíz, con dos secciones: `<head>` (información sobre la página: charset UTF-8 y title) y `<body>` (todo el contenido visible).
- Los encabezados h1 a h6 forman la jerarquía del documento: un solo h1 por página como título principal, h2 para secciones, h3 para subsecciones, sin saltarse niveles; la jerarquía ayuda al posicionamiento en buscadores y a la navegación con lectores de pantalla.
- La etiqueta `<p>` define párrafos, los bloques de texto sobre una idea; el navegador ignora los saltos de línea del código y une el texto en un flujo continuo.
- `<strong>` marca texto importante (se ve en negrita) y `<em>` marca énfasis al leer (se ve en cursiva): ambas tienen significado semántico, a diferencia de `<b>` e `<i>` que solo cambian la apariencia.
- `<mark>` resalta como marcador fluorescente, `<small>` indica información secundaria o letra chica, `<u>` subraya pero debe evitarse porque confunde con enlaces, y `<br>` fuerza un salto de línea sin ser sustituto de párrafos.
- Las etiquetas se anidan como muñecas rusas: se cierran en orden inverso al de apertura, nunca cruzadas; el navegador tolera errores pero los tolera a su manera impredecible.
- Los comentarios `<!-- -->` son notas para desarrolladores que el navegador ignora; sirven para marcar secciones y desactivar código temporalmente, nunca para guardar secretos.
- Buenas prácticas desde el día uno: minúsculas en etiquetas, cerrar todo lo que abras, indentar con consistencia, un title descriptivo por página, declarar lang y charset siempre, nombres de archivo en minúsculas sin espacios, y probar en el navegador temprano y seguido con el ciclo escribir-guardar-recargar.

- Con esta unidad ya puedes crear páginas web válidas, estructuradas y con texto bien formateado; en la siguiente unidad les agregarás enlaces e imágenes para conectarlas con el mundo.

UNIDAD 2: Texto, párrafos y navegación

En la Unidad 1 aprendiste que una página web es un documento con estructura: el doctype que le dice al navegador qué tipo de documento es, la etiqueta html que envuelve todo, el head donde vive la información invisible y el body donde aparece lo que el visitante ve en pantalla. También conociste los encabezados del h1 al h6, que son como los títulos de un libro. Esa fue la estructura, el esqueleto de la casa. Ahora, en la Unidad 2, vas a aprender a llenar esa casa de vida: el texto que la gente va a leer, los enlaces que la conectan con el resto del mundo y las imágenes que la hacen visualmente atractiva.

Esta unidad cubre tres grandes temas que van de la mano: primero, cómo organizar el texto con párrafos, saltos de línea, separadores, texto preformateado y citas; segundo, cómo crear enlaces con la etiqueta a, que es literalmente lo que hace que la web sea la web; y tercero, cómo mostrar imágenes con la etiqueta img para que tus páginas dejen de ser solo texto y empiecen a verse profesionales. Al terminar vas a poder escribir un artículo completo con párrafos bien organizados, crear un menú de navegación que conecte varias páginas, enlazar secciones dentro de la misma página y colocar imágenes con descripciones accesibles.

¿Por qué importa tanto esta unidad? Piensa en esto: la inmensa mayoría del contenido de internet es texto. Noticias, blogs, tiendas en línea, redes sociales, tutoriales como este curso, todo es texto presentado de alguna forma. Si sabes organizar texto en HTML, ya sabes presentar el noventa por ciento del contenido de la web. Y los enlaces son el pegamento que une todo: sin enlaces no existiría la navegación, no habría menús, no podrías pasar de una página a otra ni compartir contenido. La palabra web significa red, y una red sin conexiones no es red. Cada vez que haces clic en algo para ir a otro lado, hay una etiqueta a trabajando detrás. Dominarla es dominar la navegación.

¿Cómo encaja esta unidad en el curso completo? La Unidad 1 te dio el esqueleto y los títulos. Esta unidad te da el cuerpo: el contenido real que la gente consume. En las próximas unidades vas a aprender listas, tablas, formularios, audio, video y mucho más, y todo eso se va a montar sobre lo que aprendas aquí. Los párrafos que domines hoy van a contener las listas de mañana; los enlaces que crees hoy van a conectar los formularios que harás después; las imágenes que aprendas a colocar hoy van a convivir con el video de las siguientes unidades. Nada de lo que sigue funciona sin texto bien organizado y navegación sólida, así que tómate esta unidad en serio: es el piso sobre el que se construye todo lo demás.

Una cosa más antes de empezar. En esta unidad vas a notar que HTML es muy generoso: casi todo lo que quieres hacer con texto tiene una etiqueta específica. Hay etiqueta para

párrafos, para citas, para saltos de línea, para texto preformateado. Esa generosidad no es casualidad: cada etiqueta le dice al navegador y a los lectores de pantalla qué tipo de contenido es, no solo cómo se ve. Esa idea, la de que las etiquetas describen el significado y no solo la apariencia, te va a acompañar todo el curso. Apréndela desde aquí y todo lo demás te va a resultar más fácil.

Vamos por partes, con calma y con ejemplos que puedes probar tú mismo en tu editor. Abre tu archivo de práctica, escribe el código con tus propias manos y mira el resultado en el navegador. La mejor forma de aprender HTML es escribiéndolo, rompiéndolo y arreglándolo. Esta unidad es larga porque el tema lo merece: cuando la termines, vas a ver las páginas web con otros ojos, vas a entender por qué el texto se ve como se ve y vas a poder construir páginas que se leen bien y se navegan mejor. Échale ganas, que esto es de lo más útil que vas a aprender en todo el curso.

Explicación detallada

Párrafos: la unidad básica del texto

La etiqueta `p` es, sin exagerar, la etiqueta más usada de todo HTML junto con `div`. La `p` viene de paragraph, que en inglés significa párrafo, y sirve exactamente para eso: marcar un bloque de texto como un párrafo. Cada vez que el navegador encuentra una etiqueta `p` de apertura y su correspondiente cierre, sabe que ese texto forma una unidad de pensamiento completa, y automáticamente le pone un espacio antes y un espacio después para separarlo visualmente de los párrafos vecinos.

Mira este ejemplo:

```
<p>Este es un párrafo. El navegador lo muestra como un bloque de texto separado de los demás.</p>
<p>Este es otro párrafo. Fíjate en el espacio que hay entre uno y otro: ese espacio lo pone el navegador solo.</p>
```

Ese espacio automático es importante porque te ahorra trabajo: no tienes que andar agregando saltos de línea a mano entre párrafos. Pero ojo, también significa que si quieres controlar el espacio con precisión vas a necesitar CSS más adelante, porque el navegador decide cuánto espacio pone por defecto. Por ahora, quédate con la idea: un `p` es un párrafo, y los párrafos se separan solos.

Algo que confunde a muchos principiantes es qué pasa con los espacios y los saltos de línea que escribes dentro del código. Prueba esto en tu editor: escribe un párrafo con varias líneas en el código, así:

```
<p>Esta es la primera línea que escribo en el código.  
Esta es la segunda línea que escribo en el código.  
Y esta es la tercera.</p>
```

Cuando lo veas en el navegador, las tres líneas van a aparecer pegadas en una sola línea continua, como si fuera un solo renglón largo. ¿Por qué? Porque HTML ignora los saltos de línea y los espacios múltiples dentro del código: todos los espacios en blanco seguidos (espacios, tabulaciones, saltos de línea) los trata como un solo espacio. Esto es a propósito: te permite escribir tu código como quieras, con la indentación que te guste, sin que eso afecte cómo se ve la página. El navegador decide dónde cortar las líneas según el ancho de la ventana. Si achicas la ventana del navegador, el texto se reacomoda solo; si la agrandas, el texto se estira. Eso se llama flujo del texto, y es una de las bases del diseño web adaptable.

Entonces, ¿cómo decides dónde termina un párrafo y empieza otro? Con las etiquetas, no con los saltos de línea del código. Cada par de etiquetas p es un párrafo, punto. Si quieres dos párrafos, escribes dos etiquetas p. Así de simple y así de estricto. No importa cuántas líneas en blanco dejes en tu código entre un párrafo y otro: el navegador solo ve las etiquetas.

Los párrafos pueden contener otras etiquetas dentro. De hecho, casi siempre lo hacen: dentro de un p puedes poner enlaces con a, imágenes con img, texto en negritas con strong, texto en cursiva con em, y muchas más. Lo que no puedes hacer es meter un bloque dentro de otro bloque de forma incorrecta: por ejemplo, no debes meter un p dentro de otro p, ni meter un encabezado h1 dentro de un p. El párrafo es un contenedor de texto y de etiquetas en línea, no de otros bloques. Si intentas meter un p dentro de un p, el navegador lo va a interpretar de forma rara y te va a dar dolores de cabeza. Regla de oro: los párrafos contienen texto y etiquetas en línea, nada más.

Vamos a ver un ejemplo más completo, como el inicio de un artículo real:

```
<h1>Cómo cuidar tus plantas en verano</h1>  
<p>El verano es la época más difícil para las plantas de interior. El calor seca la tierra más rápido y el sol directo puede quemar las hojas.</p>  
<p>En este artículo vas a aprender tres trucos sencillos para que tus plantas sobrevivan al calor sin que tengas que estar regándolas todo el día.</p>
```


Fíjate cómo el h1 presenta el tema y los dos párrafos desarrollan la introducción. Esa es la estructura típica de cualquier artículo: título, párrafos de introducción, y luego más secciones. Cuando domines los párrafos, ya sabes escribir el cuerpo de cualquier texto en la web.

Un detalle práctico: los párrafos vacíos no sirven para dar espacio. Algunos principiantes escriben `<p></p>` varias veces para separar cosas, como si fueran saltos de línea grandes. No lo hagas: es una mala práctica porque confundes al navegador y a los lectores de pantalla, que anuncian párrafos vacíos como contenido. Si necesitas espacio, eso se controla con CSS. Los párrafos son para texto, y si no hay texto, no debe haber párrafo.

Otro detalle: la etiqueta `p` siempre debe cerrarse. En versiones viejas de HTML se permitía dejarla abierta, y algunos tutoriales antiguos todavía lo enseñan, pero en HTML5 la buena práctica es cerrarla siempre con `</p>`. Código limpio desde el principio te evita errores difíciles de encontrar después.

Para cerrar esta sección, piensa en el párrafo como la frase musical de tu página: cada `p` es una idea completa, y la secuencia de párrafos es el discurso completo. Si tus párrafos están bien organizados, con una idea por párrafo y un orden lógico, tu texto se va a leer solo. Si metes todo en un solo párrafo gigante, nadie lo va a leer, por más interesante que sea. La organización del texto no es decoración: es respeto por tu lector.

Saltos de línea y separadores: `br` y `hr`

Ya viste que HTML ignora los saltos de línea que escribes en el código. Pero a veces sí quieres forzar un salto de línea dentro de un párrafo, por ejemplo en una dirección postal, en un poema o en la letra de una canción, donde cada renglón tiene sentido por separado pero todo sigue siendo un mismo bloque de texto. Para eso existe la etiqueta `br`, que viene de *break*, romper: rompe la línea justo donde la pones.

```
<p>Lucas Ega Academy<br>
Av. de los Cursos 123<br>
Ciudad de México, México</p>
```

En el navegador eso se ve como tres renglones separados, pero sigue siendo un solo párrafo: no hay espacio extra entre renglón y renglón como lo habría entre dos párrafos. Esa es la diferencia clave entre `br` y `p`: el `br` solo baja a la siguiente línea, el `p` crea un bloque nuevo con su espacio de separación.

La etiqueta `br` es una etiqueta vacía, lo que significa que no lleva etiqueta de cierre. No existe `</br>`: solo escribes `<br>` y listo. En versiones antiguas de XHTML se escribía `<br />`, con la

diagonal al final, y todavía lo vas a ver en código viejo, pero en HTML5 lo correcto y suficiente es `<br>`. Es una de las pocas etiquetas que funcionan así, junto con `img` y `hr`, así que acostúmbrate a reconocerlas: son las que no envuelven contenido, solo hacen algo en el punto donde aparecen.

Ahora, un error muy común: usar muchos `br` seguidos para crear espacio vertical, así:

```
<p>Primer párrafo</p>
<br>
<br>
<br>
<p>Segundo párrafo</p>
```

Esto funciona visualmente, pero es una mala práctica. ¿Por qué? Porque estás usando una etiqueta que significa salto de línea para algo que no es un salto de línea: es espaciado. Es como usar cinta adhesiva para pegar un cartel en la pared cuando lo correcto es usar tachuelas: funciona, pero no es lo suyo y se nota. El espaciado entre elementos se controla con CSS, que vas a aprender más adelante en este curso. Los `br` son para saltos de línea con significado, como los renglones de un poema o de una dirección, no para empujar contenido hacia abajo.

La otra etiqueta separadora es `hr`, que viene de horizontal rule, regla horizontal. Dibuja una línea horizontal que cruza todo el ancho disponible, y se usa para marcar un cambio de tema dentro del contenido. Piensa en los libros: a veces entre una sección y otra hay una línea o un adorno que indica que el tema cambió. Eso es `hr` en la web.

```
<p>Así termina la introducción de nuestro artículo.</p>
<hr>
<p>A partir de aquí empieza el desarrollo del tema principal.</p>
```

En HTML5, la etiqueta `hr` tiene un significado semántico: representa una separación temática entre párrafos, no solo una línea decorativa. Si la quieres usar solo como adorno visual, lo correcto sería hacerlo con CSS. Pero para separar secciones de contenido relacionadas, `hr` es perfecta. También es una etiqueta vacía: se escribe `<hr>` sin cierre.

Un uso clásico de `hr` es en artículos largos: introducción, línea separadora, desarrollo, línea separadora, conclusión. Otro uso común es en páginas de contacto: la información de contacto, una línea, y luego el formulario. Te da una pausa visual que le dice al lector: ojo, aquí cambia el tema.

Vamos a juntar p, br y hr en un ejemplo completo para que veas cómo trabajan en equipo:

```
<h1>Mi poema favorito</h1>
<p>Este poema me gusta porque habla de empezar de nuevo:</p>
<p>Cada mañana es una página en blanco,<br>
cada error es una lección aprendida,<br>
cada día es una oportunidad nueva.</p>
<hr>
<p>Los poemas se disfrutan más cuando se leen en voz alta, renglón por renglón.</p>
```

Analiza lo que pasa aquí: el primer párrafo introduce, el segundo párrafo contiene el poema con br para separar los versos sin crear párrafos nuevos, el hr marca el cambio de tema, y el último párrafo cierra con un comentario. Cada etiqueta está haciendo exactamente el trabajo para el que fue creada. Cuando escribas tus propias páginas, pregúntate siempre: ¿esto es un párrafo nuevo, un salto de línea dentro del mismo bloque, o un cambio de tema? La respuesta te dice si necesitas p, br o hr.

Texto preformateado y citas: pre, blockquote, q y cite

Hay contenidos donde los espacios y los saltos de línea sí importan tal como los escribes: el código de programación es el ejemplo más claro. Si escribes código en un párrafo normal, HTML va a aplastar todos tus saltos de línea y tu indentación, y el código va a quedar ilegible. Para esos casos existe la etiqueta pre, de preformatted text, texto preformateado: todo lo que escribas dentro de pre se muestra exactamente igual, con los mismos espacios y saltos de línea.

```
<pre>
function saludar() {
    console.log('Hola, mundo');
}
saludar();
</pre>
```

El navegador muestra ese código con la indentación intacta y usando una fuente monoespaciada, es decir, una fuente donde todas las letras ocupan el mismo ancho, como las de las máquinas de escribir. Esa fuente es la tradicional para mostrar código. La etiqueta pre es indispensable cuando quieras mostrar código, arte ASCII, o cualquier texto donde el formato exacto sea parte del contenido.

Ojo con un detalle: dentro de pre, los caracteres < y > siguen siendo especiales para HTML, porque el navegador los interpreta como etiquetas. Si quieres mostrar código HTML dentro de un pre, tienes que escribir los símbolos como entidades: < para < y > para >. Así:

```
<pre>
&lt;p&gt;Este es un párrafo en HTML&lt;/p&gt;
</pre>
```

Eso se muestra en pantalla como <p>Este es un párrafo en HTML</p>, sin que el navegador lo interprete como un párrafo real. Este truco de las entidades te va a servir muchísimo cuando escribas tutoriales o documentación.

Ahora las citas. Cuando en tu texto incluyes palabras de otra persona o de otra fuente, HTML tiene etiquetas específicas para marcarlas como citas. La más importante es blockquote, que sirve para citas largas, de uno o más párrafos, que se presentan como un bloque separado. Por defecto el navegador la muestra con un margen a la izquierda, como sangría, para que se note que es una cita.

```
<p>Como decía mi abuela:</p>
<blockquote>
<p>El que madruga, Dios lo ayuda, pero el que trasnocha programando también entrega a tiempo.</p>
</blockquote>
<p>Y tenía razón, como siempre.</p>
```

Fíjate que dentro del blockquote puse un p: la cita larga es un bloque, y su contenido se organiza en párrafos normales. La etiqueta blockquote puede llevar el atributo cite con la dirección de la fuente, así: <blockquote cite='https://ejemplo.com/fuente'>. Ese atributo no se ve en pantalla, pero le dice a los navegadores y buscadores de dónde viene la cita. Es información para las máquinas, no para los ojos.

Para citas cortas, dentro de una oración, existe la etiqueta q, de quote. El navegador automáticamente le pone comillas alrededor:

```
<p>Mi profesor siempre decía <q>la práctica hace al maestro</q> antes de cada examen.</p>
```

En pantalla se ve: Mi profesor siempre decía "la práctica hace al maestro" antes de cada examen. Las comillas las pone el navegador, tú no las escribes. Esto es útil porque si algún día cambias el idioma de la página, el navegador puede adaptar el estilo de las comillas automáticamente.

Y para mencionar el título de una obra, como un libro, una película o un artículo, está la etiqueta cite. Por defecto se muestra en cursiva:

```
<p>Mi libro favorito es <cite>Cien años de soledad</cite>, de Gabriel García Márquez.</p>
```

La etiqueta cite no es para el nombre del autor, ojo: es para el título de la obra. El autor se escribe como texto normal. Esta distinción le sirve a los buscadores para entender que estás referenciando una obra creativa.

Vamos a ver un ejemplo que junta todo: un fragmento de artículo con cita larga, cita corta y referencia a una obra:

```
<h1>La importancia de leer</h1>
<p>Leer cambia la forma en que pensamos. Como escribió alguien muy sabio:</p>
<blockquote>
<p>Un lector vive mil vidas antes de morir. El que nunca lee vive solo una.</p>
</blockquote>
<p>Esa frase resume por qué insisto tanto: <q>lee todos los días, aunque sea una página</q>.</p>
<p>Si quieres empezar, te recomiendo <cite>El principito</cite>: es corto y te deja pensando.</p>
```

¿Ves cómo cada etiqueta de cita tiene su lugar? blockquote para la cita larga en bloque, q para la frase corta dentro del párrafo, cite para el título del libro. Usar la etiqueta correcta no es capricho: los lectores de pantalla anuncian las citas de forma especial, y los buscadores entienden mejor tu contenido. HTML premia a quien describe el significado, no solo la apariencia.

Un error común es usar blockquote solo para sangrar texto, porque se ve con margen a la izquierda. No lo hagas: si quieres sangría, usa CSS. blockquote significa cita, y usarlo para otra cosa confunde a las tecnologías de asistencia. La regla se repite en todo HTML: cada etiqueta tiene un significado, úsala para ese significado.

Enlaces: el puente de la web

Llegamos a la etiqueta más importante de esta unidad y, para muchos, la más importante de todo HTML: la etiqueta `a`, que viene de anchor, ancla. Con `a` y su atributo `href` creas enlaces, y los enlaces son lo que convierte un conjunto de páginas sueltas en una red, en la web. Sin la etiqueta `a` no habría navegación, no habría menús, no habría buscadores como los conocemos. Todo clic que te lleva a otro lado en internet pasa por una etiqueta `a`.

La estructura básica es así:

```
<a href='https://www.google.com'>Ir a Google</a>
```

Vamos a desarmarla pieza por pieza. La etiqueta de apertura es `<a`, luego viene el atributo `href`, que viene de hypertext reference, referencia de hipertexto: es la dirección a donde apunta el enlace. Entre comillas simples va la URL de destino. Después del signo `>` va el texto del enlace, que es lo que el usuario ve y en lo que hace clic: en este caso, Ir a Google. Y al final la etiqueta de cierre `</a>`.

El texto del enlace es importantísimo: es lo único que el visitante ve. Debe describir a dónde va a llevar el enlace. Ir a Google es un buen texto porque dice exactamente qué va a pasar. En cambio, un texto como haz clic aquí es un mal texto de enlace, porque no dice a dónde vas. Imagina una página llena de enlaces que dicen clic aquí, clic aquí, clic aquí: ¿a dónde lleva cada uno? Nadie lo sabe hasta que hace clic. Además, los lectores de pantalla suelen leer la lista de enlaces de una página por separado, y escuchar clic aquí cinco veces no le sirve a nadie. Regla de oro: el texto del enlace debe tener sentido por sí solo.

Los enlaces se muestran por defecto en color azul y subrayados, y cuando ya los visitaste se vuelven morados. Ese es el estilo clásico de la web desde sus inicios, y aunque con CSS puedes cambiarlo, es una convención tan arraigada que conviene respetarla: los usuarios ya saben que el texto azul subrayado es un enlace. Si quitas el subrayado con CSS, asegúrate de que siga siendo obvio que es un enlace.

La etiqueta `a` es una etiqueta en línea: puede vivir dentro de un párrafo, dentro de un encabezado, dentro de una lista, en casi cualquier lado donde haya texto. Mira estos usos típicos:

```
<p>Para aprender más, visita <a href='https://lucasegaacademy.com'>Lucas EGA Academy</a>, donde hay cursos gratuitos.</p>
<h2><a href='noticias.html'>Últimas noticias de tecnología</a></h2>
```

En el primer caso el enlace está dentro de un párrafo, como parte natural de la oración. En el segundo, el encabezado completo es un enlace: eso es muy común en portadas de noticias, donde el título te lleva al artículo completo. Ambas formas son válidas.

Algo que debes saber: un enlace puede contener no solo texto, sino también imágenes y otros elementos en línea. Más adelante, cuando veas imágenes, vas a poder hacer que una imagen completa sea un enlace. Por ahora quédate con la idea de que entre `<a>` y `</a>` va el contenido en el que se puede hacer clic.

Rutas absolutas y rutas relativas: las dos formas de dar una dirección

El atributo `href` necesita una dirección, y hay dos formas de escribirla: absoluta y relativa. Entender la diferencia es fundamental porque es una de las fuentes más comunes de enlaces rotos.

Una ruta absoluta es la dirección completa, como la que ves en la barra del navegador: incluye el protocolo (`https://`) y el dominio. Ejemplo:

```
<a href='https://lucasegaacademy.com/cursos/html5'>Curso de HTML5</a>
```

Esa dirección funciona desde cualquier lugar del mundo, en cualquier página, porque lleva toda la información necesaria. Es como dar tu dirección completa: calle, número, colonia, ciudad, país. No hay forma de perderse. Las rutas absolutas se usan para enlazar a sitios externos, es decir, páginas que no son tuyas o que están en otro dominio.

Una ruta relativa, en cambio, es una dirección parcial que se interpreta tomando como base la ubicación de la página actual. Ejemplo:

```
<a href='contacto.html'>Contacto</a>
```

Ese enlace dice: en la misma carpeta donde está esta página, busca el archivo `contacto.html`. Es como decirle a alguien que ya está en tu casa: la cocina está al fondo a la derecha. No necesitas darle la dirección completa de tu casa porque ya está ahí. Las rutas relativas se usan para enlazar páginas dentro de tu propio sitio.

Las rutas relativas pueden subir y bajar por las carpetas de tu sitio. Imagina esta estructura de archivos:

```
mi-sitio/  
  index.html  
  contacto.html  
  cursos/  
    html5.html  
    css.html  
  imagenes/  
    logo.png
```

Si estás en index.html y quieres enlazar a cursos/html5.html, escribes:

```
<a href='cursos/html5.html'>Curso de HTML5</a>
```

Bajas a la carpeta cursos y ahí está el archivo. Si estás en cursos/html5.html y quieres volver a index.html, que está una carpeta arriba, usas dos puntos para subir un nivel:

```
<a href='../index.html'>Inicio</a>
```

Los dos puntos.. significan sube una carpeta. Si necesitaras subir dos niveles, escribirías../../. Y para enlazar entre archivos de la misma carpeta, solo pones el nombre del archivo, como contacto.html.

¿Cuándo usar cada una? Regla práctica: usa rutas relativas para todo lo que esté dentro de tu sitio y rutas absolutas para todo lo que esté fuera. ¿Por qué? Porque las rutas relativas hacen tu sitio portable: si cambias de dominio, o si trabajas en tu computadora y luego subes los archivos al servidor, los enlaces relativos siguen funcionando sin que tengas que cambiar nada. Si hubieras usado rutas absolutas con tu dominio viejo, tendrías que actualizar cada enlace a mano. En cambio, para enlazar a Google, a YouTube o a cualquier sitio externo, la ruta absoluta es obligatoria porque no hay una base común.

Un caso especial: los enlaces que empiezan con diagonal, como . Esa diagonal inicial significa desde la raíz del sitio. Es una ruta relativa a la raíz, y es muy útil en sitios grandes porque funciona igual desde cualquier página, sin importar en qué carpeta estés. Pero ojo: solo funciona bien cuando el sitio ya está publicado en un servidor; si abres el archivo directamente en tu computadora, la raíz puede no ser la que esperas. Para tus prácticas, usa rutas relativas normales sin la diagonal inicial.

El error más común con las rutas es escribir mal el nombre del archivo: contacto.html no es lo mismo que Contacto.html ni que contacto.htm. En la mayoría de los servidores, las mayúsculas y minúsculas sí importan, y una sola letra mal escrita produce un enlace roto que

lleva a una página de error 404. Acostúmbrate a usar nombres de archivo en minúsculas, sin espacios y sin caracteres especiales: contacto.html, mis-cursos.html, sobre-mi.html. Los espacios en los nombres de archivo se convierten en códigos raros como %20 y causan problemas. Guion medio para separar palabras, minúsculas siempre, y te evitas la mitad de los enlaces rotos del mundo.

Enlaces internos con anclas: navegar dentro de la misma página

No todos los enlaces te llevan a otra página: algunos te llevan a una sección dentro de la misma página. Piensa en la Wikipedia: al inicio de un artículo largo hay un índice, y al hacer clic en cada punto saltas directamente a esa sección. Eso se hace con anclas, y es más fácil de lo que parece.

Primero le pones un identificador único a la sección de destino usando el atributo id:

```
<h2 id='capitulo-1'>Capítulo 1: Los inicios</h2>
```

El id es como el nombre de la sección, y debe ser único en toda la página: no puede haber dos elementos con el mismo id. Luego creas el enlace usando el símbolo # seguido del id:

```
<a href='#capitulo-1'>Ir al capítulo 1</a>
```

Al hacer clic, el navegador baja automáticamente hasta el elemento con ese id. El # significa en esta misma página, busca el elemento llamado así. Es una de las técnicas más útiles para páginas largas: índices, preguntas frecuentes, documentación, términos y condiciones.

Vamos a ver un ejemplo completo de página con índice interno:

```
<h1>Manual de la bicicleta</h1>
<ul>
  <li><a href='#frenos'>Cómo ajustar los frenos</a></li>
  <li><a href='#llantas'>Cómo inflar las llantas</a></li>
  <li><a href='#cadena'>Cómo lubricar la cadena</a></li>
</ul>
<h2 id='frenos'>Cómo ajustar los frenos</h2>
<p>Aquí va la explicación de los frenos...</p>
<h2 id='llantas'>Cómo inflar las llantas</h2>
<p>Aquí va la explicación de las llantas...</p>
<h2 id='cadena'>Cómo lubricar la cadena</h2>
<p>Aquí va la explicación de la cadena...</p>
```

(No te preocupes por la etiqueta ul y li: son listas, y las verás a fondo en la Unidad 3. Por ahora solo nota cómo los enlaces del índice apuntan a los id de las secciones.)

También puedes enlazar a una sección específica de otra página combinando la ruta con el ancla: `<a href='manual.html#cadena'>Cómo lubricar la cadena</a>` te lleva a la página manual.html y baja directamente a la sección de la cadena. Muy útil para conectar contenido relacionado entre páginas.

Un detalle elegante: al final de cada sección larga puedes poner un enlace de regreso arriba, así:

```
<p><a href='#arriba'>Volver arriba</a></p>
```

Y al inicio de la página pones `<body id='arriba'>` o un elemento con `id='arriba'`. Es un detalle pequeño que los lectores de páginas largas agradecen mucho.

Abrir enlaces en pestaña nueva, correo y teléfono

Por defecto, un enlace abre el destino en la misma pestaña, reemplazando la página actual. A veces quieres que se abra en una pestaña nueva, por ejemplo cuando enlazas a un sitio externo y no quieres que el visitante se vaya de tu página. Para eso sirve el atributo `target` con el valor `_blank`:

```
<a href='https://lucasegaacademy.com' target='_blank'>Lucas EGA Academy</a>
```

Con `target='_blank'`, el enlace se abre en una pestaña nueva y tu página queda abierta en la original. Suena conveniente, ¿verdad? Pues aquí va una buena práctica importante: no abuses de él. Abrir todo en pestañas nuevas es molesto: le quita al usuario el control de su navegación, y en celulares cada pestaña nueva es una interrupción. La recomendación es: usa `target='_blank'` solo para enlaces externos, y deja que los enlaces internos de tu sitio abran en la misma pestaña. Tus visitantes saben usar el botón de regresar; no decidas por ellos.

Hay un detalle de seguridad que debes conocer: cuando abres un enlace con `target='_blank'`, la página nueva puede acceder parcialmente a tu página a través de JavaScript. Para evitarlo, se acostumbra agregar el atributo `rel='noopener'`, así:

```
<a href='https://sitio-externo.com' target='_blank' rel='noopener'>Sitio externo</a>
```

El rel='noopener' corta esa conexión y protege a tus visitantes. No necesitas entender el mecanismo a fondo ahora; solo adopta la costumbre: cada vez que uses target='_blank', agrega rel='noopener'. Es una línea que te hace ver profesional y cuida a tu audiencia.

Los enlaces no solo llevan a páginas web: también pueden abrir el programa de correo o marcar un número de teléfono. Para correo se usa mailto: seguido de la dirección:

```
<a href='mailto:contacto@lucasegaacademy.com'>Escríbenos un correo</a>
```

Al hacer clic, se abre el cliente de correo del visitante con esa dirección ya puesta como destinatario. Puedes incluso prellenar el asunto así: . El%20 es un espacio codificado para URLs. Úsalo con moderación: prellenar el asunto es útil en páginas de contacto, pero no abusos.

Para teléfono se usa tel: seguido del número:

```
<a href='tel:+5215500000000'>Llámanos al 55 0000 0000</a>
```

En celulares, al hacer clic se abre la app de teléfono con el número listo para marcar. En computadoras puede abrir Skype u otra aplicación. El formato recomendado incluye el código de país con el signo +, sin espacios ni guiones en el href: +52 es México, +1 es Estados Unidos y Canadá, +34 es España. El texto visible sí puede llevar espacios para que se lea bien. Estos enlaces son oro en páginas de negocios locales: un cliente en su celular ve tu número, hace clic y ya está llamando. Cada clic que le ahorras a tu visitante es una venta que no pierdes.

Vamos a juntar todo en un ejemplo de pie de página con varios tipos de enlace:

```
<h2>Contacto</h2>
<p>Escríbenos a <a href='mailto:hola@misitio.com'>hola@misitio.com</a> o llámanos al <a href='tel:+5215500000000'>55 0000 0000</a>.</p>
<p>Síguenos en <a href='https://facebook.com/misitio' target='_blank' rel='noopener'>Facebook</a> y <a href='https://instagram.com/misitio' target='_blank' rel='noopener'>Instagram</a>.</p>
<p><a href='aviso-de-privacidad.html'>Aviso de privacidad</a> | <a href='#arriba'>Volver arriba</a></p>
```

Analízalo: correo con mailto, teléfono con tel, redes sociales externas con target='_blank' y rel='noopener', un enlace interno relativo a otra página del sitio y un ancla para volver arriba.

Seis tipos de enlace en un solo bloque, cada uno haciendo lo suyo. Cuando puedas leer un bloque así y entender cada enlace al instante, ya dominas la navegación en HTML.

Imágenes: la ventana de tu página

Una página de puro texto puede ser informativa, pero una página con imágenes es atractiva, memorable y profesional. La etiqueta `img`, de image, es la que coloca imágenes en tu página. Su estructura básica es:

```
<img src='foto.jpg' alt='Mi foto en la playa'>
```

Vamos por partes. `img` es una etiqueta vacía: no lleva etiqueta de cierre, igual que `br` y `hr`. Todo lo que necesita lo recibe a través de atributos. El atributo `src`, de source, fuente, indica la ruta del archivo de imagen: puede ser una ruta relativa como `foto.jpg` o una ruta absoluta como `https://ejemplo.com/foto.jpg`. Funciona igual que el `href` de los enlaces: si la imagen está en tu sitio, usa ruta relativa; si está en otro servidor, usa ruta absoluta.

El atributo `alt`, de alternative text, texto alternativo, es la descripción de la imagen. Y aquí va algo que debes grabarte desde hoy: el `alt` es obligatorio, siempre. No es opcional, no es decorativo, no es para después. Cada imagen de tu página debe llevar su `alt`. ¿Por qué tanta insistencia? Por tres razones de peso.

Primera: accesibilidad. Las personas con discapacidad visual navegan con lectores de pantalla, programas que leen la página en voz alta. El lector no puede ver la imagen, pero sí puede leer el `alt` y decirle al usuario qué hay ahí. Una imagen sin `alt` es, para esas personas, un hueco silencioso en tu página. Escribir un buen `alt` es incluir a todos.

Segunda: los enlaces rotos de imágenes. Si el archivo de imagen no se encuentra, por una ruta mal escrita o porque se borró del servidor, el navegador muestra en su lugar el texto del `alt`. Sin `alt`, el visitante solo ve un icono de imagen rota y no tiene idea de qué se perdió. Con un buen `alt`, al menos entiende qué imagen faltaba.

Tercera: los buscadores. Google no ve imágenes como tú: las entiende a través del `alt` y del contexto. Un `alt` descriptivo ayuda a que tus imágenes aparezcan en los resultados de búsqueda de imágenes, lo que trae visitas a tu sitio. El `alt` es, en la práctica, una pequeña pieza de posicionamiento web.

¿Cómo se escribe un buen `alt`? Describe lo que se ve en la imagen de forma concisa, como si se lo estuvieras contando a alguien por teléfono. Mal: `alt='imagen'` o `alt='foto'`. Bien: `alt='Atardecer en la playa con palmeras'`. No necesitas empezar con la foto de ni imagen de,

porque el lector de pantalla ya anuncia que es una imagen. Y no lo alargues demasiado: una o dos líneas bastan. Si la imagen es puramente decorativa, como un adorno o un separador visual que no aporta información, entonces sí se permite un alt vacío: alt="". Eso le dice al lector de pantalla: ignora esta imagen, no dice nada importante. Pero ojo: alt vacío solo para imágenes decorativas; si la imagen comunica algo, descríbela.

Los atributos width y height controlan el tamaño en que se muestra la imagen, en píxeles:

```
<img src='foto.jpg' alt='Mi foto en la playa' width='600' height='400'>
```

Con width='600' la imagen se muestra con 600 píxeles de ancho, y con height='400' con 400 de alto. Si pones solo uno de los dos, el navegador calcula el otro automáticamente para mantener las proporciones: con solo width='600', la altura se ajusta sola para que la imagen no se deforme. Eso es lo recomendado: especifica un solo lado y deja que el navegador mantenga la proporción.

¿Por qué especificar el tamaño si el navegador puede averiguarlo solo? Por el rendimiento visual: cuando el navegador sabe de antemano cuánto espacio va a ocupar la imagen, reserva ese espacio antes de que la imagen termine de descargarse. Sin width y height, el texto de la página brinca cuando la imagen aparece, porque el navegador tiene que reacomodar todo. Ese brinco es molesto y poco profesional. En HTML5 se recomienda poner width y height siempre, con las dimensiones reales de la imagen, y si quieres mostrarla a otro tamaño, lo ajustas con CSS. Así el navegador reserva el espacio correcto desde el principio.

Un detalle importante: width y height en HTML no recortan ni comprimen el archivo de imagen. Si tu foto pesa 5 megabytes y la muestras con width='300', el visitante sigue descargando los 5 megabytes completos, solo que se ven chiquitos. Para que la página cargue rápido, la imagen debe estar optimizada en su tamaño real de archivo, no solo en su tamaño de muestra. Más adelante en esta unidad verás las buenas prácticas de peso de imágenes.

Las imágenes también pueden ser enlaces: basta con poner la etiqueta img dentro de una etiqueta a:

```
<a href='https://lucasegaacademy.com'>  
  <img src='logo.png' alt='Logotipo de Lucas EGA Academy'>  
</a>
```

Al hacer clic en el logotipo, el visitante va al sitio. Esto es un patrón clásicísimo: el logo de casi todos los sitios web es un enlace a la página de inicio. Nota que el alt describe la imagen, no el destino: el lector de pantalla ya sabe que es un enlace por el contexto.

También puedes usar imágenes dentro de párrafos y encabezados, igual que los enlaces. Y puedes combinar imagen con texto descriptivo usando figure y figcaption, que verás en la siguiente sección.

Vamos con un ejemplo completo de artículo con imágenes:

```
<h1>Mi viaje a la playa</h1>
<p>El fin de semana pasado fui a la playa y tomé estas fotos:</p>
<img src='playa-amanecer.jpg' alt='Amanecer en la playa con el sol saliendo del mar' width='800' height='500'>
<p>Llegué temprano para ver el amanecer. Valió la pena madrugar:</p>
<img src='playa-palmeras.jpg' alt='Palmeras inclinadas por el viento frente al mar' width='800' height='500'>
<p>En la tarde el viento movía las palmeras y el mar se veía increíble.</p>
```

Cada imagen tiene su src, su alt descriptivo y sus dimensiones. El texto y las imágenes se alternan contando la historia juntos. Así se construyen los artículos con fotos que ves en todos los blogs del mundo.

Formatos de imagen: jpg, png, gif, webp y svg

No todas las imágenes son iguales: existen varios formatos de archivo, y cada uno sirve para algo distinto. Elegir el formato correcto es la diferencia entre una página que carga en un segundo y una que tarda diez. Vamos a ver los cinco formatos que te vas a encontrar en la web.

JPG, también escrito JPEG, es el formato de las fotografías. Comprime muy bien las imágenes con muchos colores y degradados, como fotos de personas, paisajes y productos. Su truco es la compresión con pérdida: descarta detalles que el ojo humano apenas nota para reducir mucho el peso. La desventaja es que no maneja transparencias: un JPG siempre es un rectángulo lleno. Úsalo para fotografías y cualquier imagen realista. Si tu foto se ve con cuadritos o manchas raras, la comprimiste de más: busca el equilibrio entre peso y calidad, normalmente entre 70 y 85 por ciento de calidad es el punto dulce.

PNG es el formato de los gráficos con bordes nítidos: logotipos, iconos, capturas de pantalla, dibujos con texto. Su compresión es sin pérdida, así que los bordes quedan perfectos y el texto se lee clarito. Su gran ventaja es la transparencia: un PNG puede tener fondo transparente, ideal para logotipos que van sobre fondos de colores. La desventaja es el peso: para fotografías, un PNG pesa muchísimo más que un JPG equivalente. Regla simple: foto realista, JPG; gráfico con texto o transparencia, PNG.

GIF es el formato de las animaciones cortas: esos clips chistosos que se repiten en bucle. Un GIF puede contener varios cuadros y mostrarlos en secuencia. También maneja transparencia, pero solo de un nivel: un píxel es transparente o no lo es, sin medios tonos, así que los bordes se ven dentados. Además está limitado a 256 colores, por eso las fotos en GIF se ven mal. Úsalo solo para animaciones sencillas y pequeñas. Para animaciones grandes, el video es mejor opción, como verás en unidades posteriores.

WebP es el formato moderno creado por Google: comprime mejor que JPG y que PNG, maneja transparencias y también animaciones. Una misma imagen en WebP puede pesar 30 por ciento menos que en JPG viéndose igual. Todos los navegadores modernos lo soportan. Si puedes, usa WebP para todo: es el mejor equilibrio entre calidad y peso hoy en día. La única precaución es con navegadores muy viejos, que no lo reconocen; en proyectos donde eso importe, se puede ofrecer un JPG como respaldo, pero para tus prácticas usa WebP sin miedo.

SVG es un caso aparte: no es una imagen de píxeles, sino un dibujo descrito con código, con instrucciones como dibuja un círculo aquí, una línea allá. Por eso se puede agrandar infinitamente sin perder nitidez: un logotipo en SVG se ve perfecto tanto en un celular como en una pantalla gigante. Pesa poquísimos y se puede editar con CSS. Úsalo para logotipos, iconos e ilustraciones simples. No sirve para fotografías, porque describir una foto con instrucciones de dibujo sería interminable.

Aquí tienes una guía rápida de decisión: ¿es una foto? JPG o WebP. ¿Es un logo o icono con fondo transparente? PNG o SVG. ¿Es un icono simple que debe verse nítido en cualquier tamaño? SVG. ¿Es una animación corta? GIF o WebP animado. ¿Quieres el menor peso posible con buena calidad? WebP. Con estas reglas ya decides mejor que la mayoría de los principiantes.

figure y figcaption: imágenes con pie de foto

Cuando una imagen es una unidad de contenido independiente, como una foto con su pie de foto en un artículo, HTML tiene las etiquetas `figure` y `figcaption` para marcar esa relación. `figure` envuelve la imagen y su pie; `figcaption` es el pie de foto propiamente dicho.

```
<figure>
  <img src='volcan.jpg' alt='Volcán al atardecer con nubes rosadas' width='800'
height='500'>
  <figcaption>El volcán al atardecer, fotografiado desde el mirador
norte.</figcaption>
</figure>
```

En pantalla se ve la imagen y debajo el pie de foto, normalmente en letra más pequeña. Pero lo importante no es cómo se ve, sino lo que significa: le estás diciendo al navegador que esa imagen y ese texto forman una unidad, que el pie describe a la imagen. Los lectores de pantalla y los buscadores entienden esa relación.

¿Cuál es la diferencia entre figcaption y alt? El alt es la descripción funcional de la imagen: lo que se ve, para quien no puede verla. El figcaption es información adicional para todos: el contexto, el crédito del fotógrafo, un comentario. Se complementan, no se repiten: evita escribir lo mismo en los dos. Por ejemplo, alt='Gráfica de barras con las ventas de 2024' y figcaption='Figura 1: las ventas crecieron 40 por ciento respecto a 2023. Fuente: departamento de ventas.' El alt dice qué se ve; el pie dice qué significa.

La etiqueta figure no es solo para fotos: también puede contener código, diagramas, tablas o citas con su pie. Pero su uso más común, por mucho, es imagen con pie de foto. Úsala cada vez que una imagen merezca una explicación visible.

Mira un ejemplo de galería con figure:

```
<h1>Galería del viaje</h1>
<figure>
  <img src='foto1.jpg' alt='Amanecer en la playa' width='600' height='400'>
  <figcaption>Día 1: el amanecer nos recibió con este cielo.</figcaption>
</figure>
<figure>
  <img src='foto2.jpg' alt='Mercado local con puestos de colores' width='600'
height='400'>
  <figcaption>Día 2: recorrimos el mercado local.</figcaption>
</figure>
<figure>
  <img src='foto3.jpg' alt='Cena frente al mar al atardecer' width='600'
height='400'>
  <figcaption>Día 3: cena de despedida frente al mar.</figcaption>
</figure>
```


Cada foto es una unidad con su pie, y el conjunto forma la galería. Limpio, semántico y fácil de darle estilo con CSS después. Cuando en la práctica construyas galerías, este es el patrón a seguir.

Analogías que te van a quedar grabadas

A veces un concepto técnico se entiende mejor con una imagen mental de la vida real. Aquí van las analogías clave de esta unidad: guárdalas, porque te van a servir cada vez que dudes.

El enlace es un puente. Imagina que cada página web es una isla y que internet es un archipiélago enorme. Sin puentes, cada isla está aislada: puedes verla, pero no puedes ir a ningún otro lado. La etiqueta `a` es el puente: conecta tu isla con otras islas. Algunos puentes llevan a islas lejanas de otros dueños, esos son los enlaces externos con rutas absolutas. Otros puentes conectan las distintas partes de tu propia isla, esos son los enlaces internos con rutas relativas. Y algunos puentes ni siquiera salen de tu terreno: te llevan de tu casa al jardín, esos son los enlaces con ancla dentro de la misma página. Un sitio sin enlaces es una isla sin puentes: bonita, pero inútil. Cada vez que agregues un enlace, estás construyendo un puente para que tus visitantes viajen.

La imagen es una ventana. Tu página es una habitación con paredes de texto. La etiqueta `img` abre una ventana en esa pared: a través de ella el visitante ve algo que está afuera, en otro archivo. El `src` es la dirección de lo que se ve por la ventana; si la dirección está mal, la ventana muestra un muro gris, la imagen rota. El `alt` es lo que le describes a alguien que no puede asomarse: oye, por esta ventana se ve un atardecer en la playa. Y el `width` y el `height` son el tamaño del marco de la ventana. Una ventana del tamaño correcto, con una vista clara y bien descrita, hace que la habitación se sienta viva. Una ventana rota o sin descripción es solo un hueco en la pared.

El párrafo es un vagón de tren. Tu artículo es un tren y cada párrafo es un vagón: todos van en la misma dirección, conectados en orden, y cada uno lleva su propia carga de ideas. El `br` es como una división dentro de un vagón: separa cosas sin crear un vagón nuevo. El `hr` es la estación entre tramos del viaje: aquí termina una etapa y empieza otra. Si metes todas las ideas en un solo vagón gigante, el tren se vuelve imposible de abordar; si separas cada frase en su propio vagón, el viaje se vuelve tedioso. Un vagón por idea, en buen orden, y el viaje se disfruta.

La ruta relativa es una indicación local y la absoluta es una dirección postal completa. Si estás en una fiesta y preguntas dónde está el baño, te dicen: al fondo a la derecha. Esa es una ruta relativa: solo funciona porque ya estás en la casa. Si le escribes a un amigo para que llegue a

la fiesta, le das calle, número, colonia y ciudad: esa es la ruta absoluta, funciona desde cualquier lugar. Usar una ruta absoluta para enlazar dentro de tu propio sitio es como darle tu dirección completa a alguien que ya está en tu sala: funciona, pero es torpe e innecesario. Usar una ruta relativa para enlazar a otro sitio es como decirle al fondo a la derecha a alguien que está en otra ciudad: no va a llegar nunca.

El alt es el doblaje de una película muda. Una imagen sin alt es como una escena de película sin sonido para quien no puede verla: pasa algo en pantalla, pero no sabes qué. El alt es la voz que narra la escena: por eso debe describir lo importante, no cada detalle irrelevante. Un buen narrador no dice hay un píxel azul en la esquina: dice el protagonista mira el mar al atardecer. Sé ese narrador.

Errores comunes que debes evitar

En esta sección vamos a ver los errores que cometen casi todos los principiantes con texto, enlaces e imágenes. Léelos con atención: conocer el error de antemano es la mitad de la solución.

Error 1: enlaces rotos por rutas mal escritas. Es el error número uno de la web. Escribes `<a href='contaco.html'>` cuando el archivo se llama contacto.html, o pones Contacto.html con mayúscula cuando el archivo es contacto.html en minúsculas, y el enlace lleva a una página de error. En tu computadora con Windows quizá funcione porque Windows no distingue mayúsculas, pero al subirlo al servidor, que normalmente es Linux y sí distingue, se rompe. Solución: nombres de archivo siempre en minúsculas, sin espacios, sin acentos ni eñes, palabras separadas con guion medio. Y antes de publicar, haz clic en cada enlace de tu sitio para comprobar que funciona.

Error 2: imágenes sin alt. Ya lo dijimos, pero vale repetirlo porque es el error más frecuente con imágenes: `<img src='foto.jpg'>` sin alt. Recuerda que el alt es obligatorio. Cada vez que escribas una etiqueta img, escribe el alt en el mismo momento, no lo dejes para después porque el después nunca llega. Hazlo un hábito, como ponerte el cinturón al subir al coche.

Error 3: usar br para dar espacio vertical. Varios `<br><br><br>` seguidos para separar secciones. Visualmente funciona, semánticamente es incorrecto y cuando quieras ajustar el diseño con CSS vas a sufrir. Los br son para saltos de línea con significado dentro de un bloque, como versos de un poema. El espacio entre bloques se controla con CSS.

Error 4: textos de enlace que no dicen nada. Enlaces que dicen clic aquí, entra aquí, más información sin contexto. El usuario no sabe a dónde va, y los lectores de pantalla los anuncian sin sentido. Escribe textos descriptivos: Ver el curso de HTML5, Descargar el temario

en PDF, Leer el aviso de privacidad. Si el texto del enlace no tiene sentido fuera de contexto, está mal.

Error 5: abrir todo en pestañas nuevas. Poner `target='_blank'` en cada enlace, incluso en los internos. El resultado es un usuario con veinte pestañas abiertas y sin saber cuál es cuál. Reserva `target='_blank'` para enlaces externos y deja que tu propio sitio navegue en la misma pestaña. Y cuando lo uses, no olvides `rel='noopener'`.

Error 6: imágenes pesadísimas. Subir la foto de 8 megabytes directo de la cámara y mostrarla con `width='400'`. La página tarda una eternidad en cargar, sobre todo en celulares con datos móviles, y muchos visitantes se van antes de verla. Solución: redimensiona y comprime tus imágenes antes de subirlas. Una foto para web rara vez necesita más de 1920 píxeles de ancho, y con 70 a 85 por ciento de calidad JPG se ve perfecta pesando una fracción.

Error 7: no cerrar las etiquetas p. Dejar párrafos abiertos porque en algún tutorial viejo viste que se podía. Los navegadores modernos intentan adivinar dónde termina el párrafo, pero a veces adivinan mal y el diseño se rompe de formas misteriosas. Cierra siempre tus etiquetas: `<p>texto</p>`. Sin excepciones.

Error 8: meter bloques dentro de párrafos. Cosas como `<p><h2>Título</h2></p>` o `<p><p>texto</p></p>`. El párrafo solo puede contener texto y etiquetas en línea. Si metes un bloque dentro, el navegador cierra el párrafo antes de tiempo y tu estructura queda rota. Los encabezados, las listas y los blockquote van fuera de los párrafos, como hermanos, no como hijos.

Error 9: usar blockquote para sangrar texto. Como el blockquote se ve con margen a la izquierda, algunos lo usan para mover texto a la derecha sin que sea una cita. Es un abuso semántico: confunde a los lectores de pantalla y ensucia el código. La sangría se hace con CSS; blockquote es solo para citas.

Error 10: olvidar el # en los enlaces internos. Escribir `<a href='seccion-1'>` en lugar de `<a href='#seccion-1'>`. Sin el #, el navegador busca un archivo llamado seccion-1 y no lo encuentra. El # es lo que indica que es un id dentro de la página actual. Detalle pequeño, error frecuente: revísalo siempre.

Error 11: id duplicados. Poner el mismo id en dos secciones y luego enlazar a él: el navegador salta al primero que encuentra y el segundo queda inalcanzable por ancla. Los id deben ser únicos en toda la página, como los números de teléfono: si dos personas tuvieran el mismo número, las llamadas llegarían a la persona equivocada.

Error 12: rutas de imagen absolutas a tu computadora. Escribir `src='C:/Usuarios/TuNombre/Imágenes/foto.jpg'` funciona en tu máquina y en ninguna otra. Al subir el sitio, esas imágenes se rompen. Usa rutas relativas para las imágenes de tu sitio, igual que con los enlaces internos, y organiza tus archivos en carpetas claras.

Buenas prácticas: el manual del profesional

Ya viste los errores; ahora las costumbres de quien hace las cosas bien. Adoptar estas prácticas desde el principio te va a ahorrar años de malos hábitos.

Primera: texto alternativo descriptivo siempre. No solo pongas el alt para cumplir: escríbelo pensando en la persona que lo va a escuchar. Describe lo relevante, sé conciso, no repitas información que ya está en el texto cercano. Si la imagen es un gráfico con datos, el alt debe transmitir la conclusión del gráfico, no solo decir que es un gráfico. Ejemplo: en lugar de `alt='gráfica'`, escribe `alt='Gráfica de barras que muestra un crecimiento de ventas del 40 por ciento en 2024'`.

Segunda: no abras todo en pestaña nueva. Ya lo mencionamos en los errores, pero como práctica positiva suena así: respeta el control del usuario sobre su navegación. Enlaces internos en la misma pestaña, enlaces externos en pestaña nueva con `rel='noopener'`. Tus visitantes te lo van a agradecer con más tiempo en tu sitio.

Tercera: optimiza el peso de tus imágenes. Antes de subir una imagen, pregúntate: ¿qué tamaño real necesita? Rediménsionala a ese tamaño, comprímela al 70-85 por ciento si es JPG, y considera WebP para un peso aún menor. Una página cuyas imágenes pesen en total menos de 1 megabyte carga rápido hasta con conexiones lentas. La velocidad de carga no es un lujo: es parte de la experiencia, y los buscadores la toman en cuenta para posicionarte.

Cuarta: nombres de archivo limpios y descriptivos. En lugar de `IMG_20240919_183452.jpg`, usa `atardecer-playa-cancun.jpg`. Los nombres descriptivos te ayudan a encontrar tus archivos, ayudan a los buscadores a entender tus imágenes y evitan los problemas de espacios y caracteres especiales. Minúsculas, guiones medios, sin acentos: esa es la fórmula.

Quinta: un párrafo, una idea. No escribas párrafos de veinte líneas que mezclan tres temas. Los párrafos cortos, de tres a cinco líneas, se leen mejor en pantalla, donde la gente escanea en lugar de leer palabra por palabra. Si un párrafo se está alargando, casi siempre se puede partir en dos.

Sexta: jerarquía clara con encabezados y párrafos. Usa los h1 a h6 de la Unidad 1 para titular las secciones y los p de esta unidad para desarrollarlas. Un documento bien jerarquizado se

puede leer solo con los títulos, como un mapa. Si tus títulos cuentan la historia por sí solos, tu estructura es buena.

Séptima: verifica tus enlaces periódicamente. Los sitios externos cambian, las páginas se mueven, los dominios caducan. Un enlace que funcionaba hace un año puede estar roto hoy. Si tu sitio tiene muchos enlaces externos, revísalos cada cierto tiempo. Un sitio lleno de enlaces rotos pierde credibilidad rápido.

Octava: piensa en el lector de pantalla aunque no lo uses. Cada vez que escribas una imagen, pregúntate qué escucharía una persona ciega. Cada vez que escribas un enlace, pregúntate si su texto tiene sentido leído en una lista. La accesibilidad no es un extra para después: es parte de escribir buen HTML desde el día uno. Y tiene un beneficio colateral: lo que es bueno para los lectores de pantalla suele ser bueno para los buscadores.

Novena: separa contenido de presentación. Las etiquetas HTML describen qué es el contenido; la apariencia se controla con CSS. Cada vez que sientas la tentación de usar una etiqueta para lograr un efecto visual, como `br` para espaciar o `blockquote` para sangrar, detente: esa es chamba de CSS. HTML para el qué, CSS para el cómo. Esta disciplina te va a hacer la vida mucho más fácil cuando llegues a la parte de estilos.

Décima: prueba en el navegador mientras escribes. No escribas cincuenta líneas de HTML a ciegas y luego veas qué pasó: escribe un bloque, guárdalo, recarga el navegador y mira el resultado. El ciclo escribir, guardar, recargar, observar es tu mejor maestro. Los errores se encuentran más fácil cuando el código es corto y fresco en tu memoria.

Ejemplos paso a paso

La teoría ya la tienes; ahora vamos a verla en acción con cuatro ejemplos completos, comentados línea por línea. Te recomiendo que copies cada ejemplo en tu editor, lo guardes y lo abras en el navegador para ver el resultado con tus propios ojos. Leer código sirve, pero escribirlo y verlo funcionar es lo que realmente fija el conocimiento.

Ejemplo 1: artículo con párrafos y citas

Vamos a construir el inicio de un artículo de blog sobre la lectura, usando párrafos, una cita larga con `blockquote`, una cita corta con `q` y la referencia a un libro con `cite`. Este es el patrón de cualquier artículo editorial en la web.

```

<!-- Título principal del artículo -->
<h1>Por qué deberías leer todos los días</h1>

<!-- Párrafo de introducción: presenta el tema -->
<p>Leer es uno de los hábitos más poderosos que puedes desarrollar. No necesitas leer un libro por semana: con veinte minutos al día es suficiente para notar la diferencia en tu vocabulario, tu concentración y tu forma de pensar.</p>

<!-- Segundo párrafo: desarrolla la idea con un dato -->
<p>Los estudios muestran que las personas que leen con regularidad manejan mejor el estrés y toman decisiones más reflexivas. La lectura es como ejercicio para el cerebro: entre más la practicas, más fuerte se vuelve.</p>

<!-- Línea separadora: aquí cambia el tema, de los beneficios a la cita -->
<hr>

<!-- Párrafo que introduce la cita larga -->
<p>El escritor lo dijo mejor que nadie:</p>

<!-- blockquote: cita larga en bloque, con su fuente en el atributo cite -->
<blockquote cite='https://ejemplo.com/fuente'>
<p>Un lector vive mil vidas antes de morir. El que nunca lee vive solo una.</p>
</blockquote>

<!-- Párrafo con cita corta usando q: el navegador pone las comillas -->
<p>Por eso mi consejo es sencillo: <q>lee todos los días, aunque sea una página</q>. La constancia le gana a la intensidad.</p>

<!-- Párrafo final con referencia a una obra usando cite -->
<p>Si no sabes por dónde empezar, prueba con <cite>El principito</cite>: es corto, es profundo y te deja pensando por días.</p>

```

Analicemos las decisiones. Los dos primeros párrafos forman la introducción con los beneficios. El hr marca el cambio de tema: pasamos de los beneficios a la inspiración. El blockquote contiene la cita larga dentro de su propio p, y lleva el atributo cite con la fuente para las máquinas. La etiqueta q envuelve la frase corta dentro del párrafo y el navegador le pone las comillas automáticamente. Y cite marca el título del libro en cursiva. Cada etiqueta está describiendo el significado de su contenido, no solo su apariencia. Si mañana le aplicas CSS a este artículo, podrás darle a las citas un estilo elegante sin tocar el HTML, porque la estructura ya dice qué es cada cosa.

Ejemplo 2: menú de navegación con enlaces

Todo sitio web necesita un menú: la barra de enlaces que te lleva a las secciones principales. Vamos a construir uno con una lista de enlaces, usando rutas relativas para las páginas internas y una ruta absoluta para el blog externo. Las etiquetas `ul` y `li` son listas: las estudiarás a fondo en la Unidad 3, pero aquí las usamos de forma básica porque los menús se construyen con listas.

```
<!-- nav indica que este bloque es la navegación principal del sitio -->
<nav>
  <!-- ul: lista sin orden, cada li es un elemento del menú -->
  <ul>
    <!-- Enlace interno con ruta relativa: misma carpeta -->
    <li><a href='index.html'>Inicio</a></li>
    <!-- Enlace interno con ruta relativa: archivo en la misma carpeta -->
    <li><a href='cursos.html'>Cursos</a></li>
    <!-- Enlace interno con ruta relativa: baja a la carpeta blog -->
    <li><a href='blog/articulos.html'>Blog</a></li>
    <!-- Enlace interno con ruta relativa: página de contacto -->
    <li><a href='contacto.html'>Contacto</a></li>
    <!-- Enlace externo con ruta absoluta: se abre en pestaña nueva -->
    <li><a href='https://lucasegaacademy.com' target='_blank' rel='noopener'>Lucas
EGA Academy</a></li>
  </ul>
</nav>
```

Fíjate en los detalles profesionales. La etiqueta `nav` envuelve el menú y le dice al navegador: esto es navegación, no contenido cualquiera. Los textos de los enlaces son descriptivos: Inicio, Cursos, Blog, Contacto. Las rutas internas son relativas, así el menú funciona igual en tu computadora y en el servidor sin cambios. El enlace externo usa ruta absoluta, `target='_blank'` para no sacar al visitante de tu sitio, y `rel='noopener'` por seguridad. Este menú lo puedes copiar al inicio de cada página de tu sitio y tendrás navegación consistente en todo el proyecto. Cuando aprendas CSS, le darás estilo de barra horizontal con colores, pero la estructura ya está lista y es correcta desde hoy.

Ejemplo 3: galería de imágenes con figure

Vamos a armar una galería de fotos de un viaje, donde cada foto es una unidad con su pie descriptivo. Usaremos `figure` y `figcaption`, con `alt` descriptivos y dimensiones declaradas.

```

<!-- Título de la galería -->
<h1>Galería: mi viaje a Oaxaca</h1>

<!-- Párrafo introductorio -->
<p>Estos son mis momentos favoritos del viaje. Cada foto cuenta una historia.</p>

<!-- Primera foto: figure agrupa imagen y pie como una unidad -->
<figure>
  <!-- img con ruta relativa, alt descriptivo y dimensiones reales -->
  <img src='imagenes/zocalo.jpg' alt='Zócalo de Oaxaca al atardecer con la catedral
iluminada' width='800' height='500'>
  <!-- figcaption: pie de foto con contexto adicional, no repite el alt -->
  <figcaption>Día 1: el zócalo al atardecer, con la catedral de fondo.</figcaption>
</figure>

<!-- Segunda foto -->
<figure>
  <img src='imagenes/monte-alban.jpg' alt='Ruinas de Monte Albán con pirámides bajo
un cielo azul' width='800' height='500'>
  <figcaption>Día 2: las ruinas de Monte Albán, impresionantes en
persona.</figcaption>
</figure>

<!-- Tercera foto -->
<figure>
  <img src='imagenes/mezcal.jpg' alt='Artesano sirviendo mezcal en jícara de barro'
width='800' height='500'>
  <figcaption>Día 3: aprendimos cómo se hace el mezcal artesanal.</figcaption>
</figure>

<!-- Párrafo de cierre -->
<p>¿Cuál fue tu favorita? Escríbeme a <a
href='mailto:hola@misitio.com'>hola@misitio.com</a> y cuéntame.</p>

```

Observa las buenas prácticas aplicadas. Las imágenes están en la carpeta imágenes, con rutas relativas: si mueves todo el sitio, la galería sigue funcionando. Los nombres de archivo son descriptivos y en minúsculas: zocalo.jpg, monte-alban.jpg. Cada alt describe lo que se ve sin repetir el pie: el alt dice zócalo al atardecer con la catedral iluminada y el pie agrega el contexto del día 1. Los width y height reservan el espacio para que la página no brinque al cargar. Y el cierre combina un párrafo con un enlace mailto, juntando lo aprendido. Esta estructura de galería es la base de portafolios, tiendas en línea y blogs de viajes: la vas a usar cientos de veces.

Ejemplo 4: página con anclas internas

Para el último ejemplo vamos a construir una página de preguntas frecuentes, donde el índice de arriba salta a cada respuesta con anclas. Es el uso más práctico de los id y el símbolo #.

```

<!-- El body lleva un id para poder volver arriba desde cualquier punto -->
<body id='arriba'>

<!-- Título de la página -->
<h1>Preguntas frecuentes</h1>

<!-- Índice con enlaces internos: cada href apunta al id de su sección -->
<p><a href='#envios'>¿Cuánto tardan los envíos?</a></p>
<p><a href='#pagos'>¿Qué formas de pago aceptan?</a></p>
<p><a href='#devoluciones'>¿Puedo devolver un producto?</a></p>

<!-- Línea separadora entre el índice y el contenido -->
<hr>

<!-- Sección 1: el h2 lleva el id que coincide con el enlace del índice -->
<h2 id='envios'>¿Cuánto tardan los envíos?</h2>
<p>Los envíos tardan de 3 a 5 días hábiles dentro del país. Te enviamos un correo con el número de guía en cuanto tu pedido sale del almacén.</p>
<!-- Enlace de regreso arriba al final de cada sección -->
<p><a href='#arriba'>Volver arriba</a></p>

<hr>

<!-- Sección 2 -->
<h2 id='pagos'>¿Qué formas de pago aceptan?</h2>
<p>Aceptamos tarjetas de crédito y débito, transferencias bancarias y pagos en tiendas de conveniencia. Todos los pagos con tarjeta pasan por una conexión segura.</p>
<p><a href='#arriba'>Volver arriba</a></p>

<hr>

<!-- Sección 3 -->
<h2 id='devoluciones'>¿Puedo devolver un producto?</h2>
<p>Sí. Tienes 30 días naturales para devolver cualquier producto sin usar. Escríbenos a <a href='mailto:ayuda@misitio.com'>ayuda@misitio.com</a> y te explicamos el proceso paso a paso.</p>
<p><a href='#arriba'>Volver arriba</a></p>

</body>

```

Este ejemplo junta casi todo lo de la unidad: párrafos organizados, hr como separadores temáticos, enlaces internos con #, id únicos en cada sección, enlaces de regreso arriba, y un mailto en la última respuesta. Prueba este código en tu navegador y haz clic en las preguntas:

vas a ver cómo la página salta suavemente a cada sección. Esa es la magia de las anclas, y es la técnica detrás de los índices de Wikipedia, las documentaciones técnicas y las páginas de preguntas frecuentes de todo internet. Nota también que los id usan palabras en minúsculas separadas con guiones: envios, pagos, devoluciones. Sin espacios, sin acentos, fáciles de escribir en el href sin equivocarte.

Práctica

Llegó la hora de ensuciarte las manos. Aquí tienes cinco ejercicios que cubren todo lo de la unidad: párrafos, citas, enlaces de todos los tipos e imágenes. Haz cada ejercicio primero por tu cuenta, sin mirar la solución, y luego compara. Equivocarte es parte del proceso: cada error que corriges es una lección que no se te olvida.

Ejercicio 1: artículo con párrafos, separador y cita

Enunciado: escribe el código HTML de un artículo titulado Mis propósitos de año nuevo que tenga: un h1 con el título, tres párrafos (uno de introducción, uno con los propósitos y uno de cierre), una línea hr entre el segundo y el tercer párrafo para marcar el cambio de tema, y una cita corta con q dentro del segundo párrafo con la frase: el éxito es la suma de pequeños esfuerzos. No uses CSS, solo HTML.

Solución paso a paso:

Paso 1: abre con el h1 para el título del artículo. El h1 es el encabezado principal y solo debe haber uno por página.

Paso 2: escribe el primer párrafo con la etiqueta p: la introducción donde cuentas por qué haces propósitos cada año.

Paso 3: escribe el segundo párrafo con tus propósitos, e incluye la frase de la cita envuelta en la etiqueta q. Recuerda: tú no escribes las comillas, las pone el navegador.

Paso 4: coloca la etiqueta hr sola, sin cierre, para separar el desarrollo del cierre.

Paso 5: escribe el tercer párrafo con tu reflexión final y cierra todas las etiquetas p que abriste.

El código queda así:

```
<h1>Mis propósitos de año nuevo</h1>
<p>Cada enero me siento a escribir mis propósitos del año. No siempre los cumplo
todos, pero el simple hecho de escribirlos me ayuda a tener más claro lo que quiero
lograr.</p>
<p>Este año mis propósitos son aprender HTML, leer doce libros y hacer ejercicio
tres veces por semana. Como dice el dicho, <q>el éxito es la suma de pequeños
esfuerzos</q>, así que voy a enfocarme en avanzar poco a poco cada día.</p>
<hr>
<p>Sé que habrá días difíciles, pero también sé que en diciembre voy a estar
orgulloso de lo avanzado. Este es mi año.</p>
```

Verifica tu solución: ¿tienes un solo h1? ¿Los tres párrafos están cerrados con </p>? ¿La etiqueta q envuelve solo la frase citada? ¿El hr está entre el segundo y el tercer párrafo? Si todo eso se cumple, lo hiciste bien.

Ejercicio 2: poema con saltos de línea y texto preformateado

Enunciado: crea una página con un h1 titulado Rincón de poesía y código que contenga: un párrafo con un poema de tres versos usando br para separar los versos sin crear párrafos nuevos, un hr, y luego un bloque pre con tres líneas de código HTML de ejemplo donde muestres cómo se escribe un párrafo (tendrás que usar las entidades < y > para que los símbolos se vean como texto).

Solución paso a paso:

Paso 1: escribe el h1 con el título.

Paso 2: abre un p, escribe el primer verso, pon
, escribe el segundo verso, pon
, escribe el tercer verso y cierra el p. Todo dentro del mismo párrafo.

Paso 3: agrega <hr> para separar la poesía del código.

Paso 4: abre un pre y dentro escribe las tres líneas de código usando < en lugar de < y > en lugar de >. Respeta la indentación: el pre la conserva tal cual.

Paso 5: cierra el pre y revisa en el navegador que el poema se vea en tres renglones y el código con su indentación.

El código queda así:

```
<h1>Rincón de poesía y código</h1>
<p>La luna sale a pasear,<br>
las estrellas la acompañan,<br>
y la noche empieza a cantar.</p>
<hr>
<pre>
<h1>Mi primer título</h1>;
<p>Mi primer párrafo con texto.</p>;
<a href='ejemplo.html'>Mi primer enlace</a>;
</pre>
```

Verifica tu solución: ¿los tres versos están dentro de un solo p? ¿Usaste
 sin etiqueta de cierre? ¿El código dentro del pre se muestra como texto y no se interpreta como HTML? Si en el navegador ves literalmente <h1>Mi primer título</h1> como texto, las entidades funcionaron.

Ejercicio 3: menú de navegación completo

Enunciado: construye el menú de navegación de un sitio con estas páginas: inicio (index.html), nosotros (nosotros.html), servicios (servicios.html), blog externo (<https://blog.misitio.com>) y contacto (contacto.html). Requisitos: usa la etiqueta nav, cada enlace debe tener un texto descriptivo, los enlaces internos abren en la misma pestaña, el enlace externo abre en pestaña nueva con rel='noopener', e incluye al final un enlace con ancla que diga Volver arriba apuntando a #arriba.

Solución paso a paso:

Paso 1: abre la etiqueta nav para indicar que es navegación.

Paso 2: agrega los cuatro enlaces internos con rutas relativas: index.html, nosotros.html, servicios.html y contacto.html. Sin target, porque los internos abren en la misma pestaña por defecto.

Paso 3: agrega el enlace externo con la ruta absoluta <https://blog.misitio.com>, más target='_blank' y rel='noopener'.

Paso 4: agrega el enlace Volver arriba. Recuerda que para que funcione, alguna parte de tu página debe tener id='arriba', normalmente el body.

Paso 5: cierra el nav y comprueba cada enlace haciendo clic.

El código queda así:

```
<nav>
  <p>
    <a href='index.html'>Inicio</a> |
    <a href='nosotros.html'>Nosotros</a> |
    <a href='servicios.html'>Servicios</a> |
    <a href='https://blog.misitio.com' target='_blank' rel='noopener'>Blog</a> |
    <a href='contacto.html'>Contacto</a> |
    <a href='#arriba'>Volver arriba</a>
  </p>
</nav>
```

Verifica tu solución: ¿los internos no tienen target? ¿El externo tiene target='_blank' y rel='noopener'? ¿El enlace de arriba lleva el #? ¿Los textos describen el destino? Un error típico aquí es olvidar el # o ponerle target='_blank' a los internos: revísalo.

Ejercicio 4: galería de imágenes con pies de foto

Enunciado: crea una galería de tres fotos de tu ciudad con la siguiente estructura por cada foto: un figure que contenga un img con src relativo a la carpeta fotos, alt descriptivo de lo que se ve, width='700' y height='450', y un figcaption con un pie que aporte contexto sin repetir el alt. Agrega un h1 con el título Mi ciudad en fotos y un párrafo de introducción. Al final, agrega un párrafo con un enlace a tu correo con mailto para que te envíen sus fotos favoritas.

Solución paso a paso:

Paso 1: escribe el h1 y el párrafo de introducción.

Paso 2: por cada foto, abre figure, escribe el img con todos sus atributos (src, alt, width, height), luego el figcaption con el contexto, y cierra figure. Repite tres veces.

Paso 3: al escribir cada alt, describe lo que se ve en la foto como si se lo contaras a alguien por teléfono. Al escribir cada figcaption, agrega información nueva: el lugar, la hora, una anécdota.

Paso 4: cierra con un párrafo que incluya el enlace mailto a tu correo.

El código queda así:

```

<h1>Mi ciudad en fotos</h1>
<p>Estos son mis lugares favoritos de la ciudad donde vivo. Todas las fotos las
tomé yo.</p>
<figure>
  <img src='fotos/plaza.jpg' alt='Plaza principal con jardines y bancas al
atardecer' width='700' height='450'>
  <figcaption>La plaza principal, mi lugar favorito para leer los
domingos.</figcaption>
</figure>
<figure>
  <img src='fotos/mercado.jpg' alt='Pasillo del mercado con puestos de frutas de
colores' width='700' height='450'>
  <figcaption>El mercado de la Merced: aquí compro la fruta más
fresca.</figcaption>
</figure>
<figure>
  <img src='fotos/malecon.jpg' alt='Malecón con palmeras y el mar al fondo en la
mañana' width='700' height='450'>
  <figcaption>El malecón a las siete de la mañana, antes de que llegue la
gente.</figcaption>
</figure>
<p>¿Tienes fotos de tu ciudad? Mándamelas a <a
href='mailto:yo@misitio.com'>yo@misitio.com</a>.</p>

```

Verifica tu solución: ¿cada img tiene su alt? ¿El alt y el figcaption dicen cosas distintas? ¿Las rutas son relativas a la carpeta fotos? ¿Los width y height están declarados? ¿El mailto lleva tu correo real? Si cambias las fotos por las tuyas, esta galería ya es publicable.

Ejercicio 5: página de receta con todo lo aprendido

Enunciado: este es el ejercicio integrador: crea la página de una receta de cocina que use todo lo de la unidad. Debe tener: h1 con el nombre de la receta, un párrafo de introducción, una imagen del platillo con figure, figcaption, alt descriptivo y dimensiones, un índice con enlaces internos a tres secciones (ingredientes, preparación y consejos) usando id y #, las tres secciones con sus h2 e id correspondientes, un blockquote con un consejo de un chef famoso incluyendo su nombre con texto normal y el título de su libro con cite, enlaces de Volver arriba al final de cada sección separados por hr, y al final un párrafo con un enlace externo a un sitio de cocina que abra en pestaña nueva con rel='noopener'.

Solución paso a paso:

Paso 1: escribe el h1 y el párrafo de introducción que antoje.

Paso 2: agrega el figure con la imagen del platillo: src relativo, alt que describa el platillo, width y height, y un figcaption con contexto.

Paso 3: crea el índice con tres enlaces internos: #ingredientes, #preparacion y #consejos. No olvides el #.

Paso 4: escribe las tres secciones: cada una con su h2 que lleve el id correspondiente y sus párrafos de contenido. En la sección de consejos incluye el blockquote con el consejo del chef y la referencia a su libro con cite.

Paso 5: al final de cada sección agrega el enlace Volver arriba y separa las secciones con hr.

Paso 6: cierra con el párrafo y el enlace externo con target='_blank' y rel='noopener'.
Asegúrate de que el body tenga id='arriba'.

El código queda así:

```
<body id='arriba'>
<h1>Tacos al pastor caseros</h1>
<p>Los tacos al pastor son el rey de la comida callejera mexicana. Esta receta te
enseña a prepararlos en casa con ingredientes fáciles de conseguir y un sabor que
compite con cualquier taquería.</p>
<figure>
  <img src='fotos/tacos.jpg' alt='Tres tacos al pastor con piña, cebolla y cilantro en
plato de barro' width='700' height='450'>
  <figcaption>Así deben verse tus tacos al final: jugosos, con piña dorada
encima.</figcaption>
</figure>
<p><a href='#ingredientes'>Ingredientes</a> | <a href='#preparacion'>Preparación</a> |
<a href='#consejos'>Consejos</a></p>
<hr>
<h2 id='ingredientes'>Ingredientes</h2>
<p>Para la carne: 1 kilo de lomo de cerdo en bisteces delgados, 3 chiles guajillo, 1
chile ancho, media taza de jugo de naranja, un cuarto de taza de vinagre blanco, 2
dientes de ajo y sal al gusto.</p>
<p>Para servir: tortillas de maíz, piña en rebanadas, cebolla picada, cilantro,
limones y tu salsa favorita.</p>
<p><a href='#arriba'>Volver arriba</a></p>
<hr>
<h2 id='preparacion'>Preparación</h2>
<p>Remoja los chiles en agua caliente, lícuales con el jugo de naranja, el vinagre, el
ajo y la sal. Marina la carne en esta mezcla por al menos 4 horas en el
refrigerador.</p>
<p>Cocina la carne en un sartén muy caliente hasta que se dore por ambos lados. Pica
la carne, calienta las tortillas y arma tus tacos con piña asada, cebolla y
```



```

cilantro.</p>
<p><a href='#arriba'>Volver arriba</a></p>
<hr>
<h2 id='consejos'>Consejos</h2>
<p>El secreto está en el marinado: entre más tiempo repose la carne, más sabor agarra. Si puedes, marínala toda la noche.</p>
<blockquote>
<p>La cocina mexicana es de paciencia: los sabores necesitan tiempo para conocerse.</p>
</blockquote>
<p>Lo dijo el chef Ricardo Muñoz Zurita, autor de <cite>Diccionario enciclopédico de la gastronomía mexicana</cite>, y tiene toda la razón.</p>
<p><a href='#arriba'>Volver arriba</a></p>
<hr>
<p>¿Quieres más recetas? Visita <a href='https://ejemplo-cocina.com' target='_blank' rel='noopener'>este sitio de cocina mexicana</a> con cientos de recetas tradicionales.</p>
</body>

```

Verifica tu solución: recorre la lista completa. ¿H1 único? ¿Imagen con alt, dimensiones y pie? ¿Índice con # y secciones con id que coinciden exactamente? ¿Blockquote con la cita y cite para el libro? ¿Enlaces de regreso arriba funcionando? ¿hr separando secciones? ¿Enlace externo con target y rel? Si todo está en su lugar, acabas de construir una página completa y profesional usando solo lo aprendido en esta unidad. Eso es un gran logro: guárdala, porque en las siguientes unidades le vas a agregar listas, tablas y estilos.

Resumen de la unidad

- La etiqueta p marca un párrafo: un bloque de texto con una idea, separado automáticamente de los demás por el navegador.
- HTML ignora los saltos de línea y espacios múltiples del código: la estructura la definen las etiquetas, no el formato del texto fuente.
- La etiqueta br fuerza un salto de línea dentro de un bloque, sin crear un párrafo nuevo; es una etiqueta vacía y no debe usarse para dar espacio vertical.
- La etiqueta hr dibuja una línea que marca un cambio de tema entre secciones; también es una etiqueta vacía con significado semántico.
- La etiqueta pre conserva espacios y saltos de línea tal cual, ideal para mostrar código con su indentación intacta.

- Para citas largas en bloque usa blockquote, para citas cortas dentro de un párrafo usa q, y para títulos de obras usa cite.
- La etiqueta a crea enlaces con el atributo href: el texto del enlace debe describir el destino y tener sentido por sí solo.
- Las rutas absolutas llevan la dirección completa y se usan para sitios externos; las rutas relativas parten de la página actual y se usan dentro de tu propio sitio.
- Con el símbolo # y el atributo id creas enlaces internos que saltan a secciones de la misma página; los id deben ser únicos.
- El atributo target='_blank' abre el enlace en pestaña nueva: úsalo solo para enlaces externos y acompáñalo siempre de rel='noopener'.
- Los enlaces mailto: abren el correo y los enlaces tel: marcan números de teléfono, muy útiles en páginas de contacto y negocios locales.
- La etiqueta img muestra imágenes con src para la ruta y alt para la descripción: el alt es obligatorio siempre por accesibilidad, posicionamiento y como respaldo si la imagen falla.
- Los atributos width y height reservan el espacio de la imagen y evitan que la página brinque al cargar; especifica uno y deja que el navegador calcule el otro para no deformar.
- Elige el formato de imagen según el caso: JPG o WebP para fotos, PNG o SVG para gráficos con transparencia y bordes nítidos, GIF para animaciones cortas, SVG para iconos y logotipos escalables.
- Las etiquetas figure y figcaption agrupan una imagen con su pie de foto como una unidad de contenido con significado.
- Buenas prácticas de la unidad: alt descriptivo siempre, no abrir todo en pestaña nueva, optimizar el peso de las imágenes, nombres de archivo en minúsculas sin espacios, un párrafo por idea y probar en el navegador mientras escribes.
- Analogías para recordar: el enlace es un puente entre islas, la imagen es una ventana con su descripción para quien no puede asomarse, y el párrafo es un vagón del tren de tu artículo.

UNIDAD 3: Listas, tablas y contenedores

Introducción

Hasta este momento ya tienes las bases del HTML5 bien plantadas: sabes qué es un documento HTML, cómo se estructura con `html`, `head` y `body`, qué son las etiquetas y cómo se anidan, y ya trabajaste con los elementos de texto como párrafos, encabezados, negritas, cursivas y enlaces. Eso ya te permite construir páginas sencillas que comunican una idea. Pero una página web de verdad casi nunca es solo un montón de párrafos sueltos: la información se organiza, se agrupa, se ordena y se presenta en formas visuales que el lector entiende de un vistazo.

Piensa en cualquier página que visites todos los días: el menú de navegación que está arriba con los enlaces principales, la lista de ingredientes de una receta, una tabla con precios de un servicio, las tarjetas de un catálogo de productos, una sección de preguntas frecuentes. Todo eso que ves y usas sin pensar está construido con los elementos que vas a dominar en esta unidad: listas, tablas y contenedores.

Esta unidad cubre las clases 7, 8 y 9 del curso. En la clase 7 aprendes las listas: `ul` para listas no ordenadas con viñetas, `ol` para listas ordenadas con números, y `li` para cada elemento de la lista. En la clase 8 aprendes las tablas: `table` como contenedor, `tr` para cada fila, `th` para los encabezados y `td` para los datos, además de `caption` para darle un título descriptivo a la tabla. Y en la clase 9 aprendes los dos contenedores genéricos más importantes de todo el HTML: `div`, que es un bloque contenedor, y `span`, que es un fragmento de texto en línea. Con estos tres grupos de etiquetas das un salto enorme: pasas de escribir contenido suelto a organizar información como un profesional.

¿Por qué importa tanto esta unidad? Porque el HTML no es solo un lenguaje para escribir texto bonito, es un lenguaje para estructurar información. Y estructurar información es exactamente lo que hacen las listas, las tablas y los contenedores. Una lista le dice al navegador y al lector: estos elementos pertenecen al mismo grupo y están al mismo nivel. Una tabla le dice: estos datos tienen una relación de filas y columnas, y se entienden mejor si los ves alineados. Un `div` le dice: todo esto que está adentro forma una sección, una tarjeta, un bloque con sentido propio. Cuando dominas estos tres conceptos, dejas de hacer páginas que parecen documentos de texto y empiezas a hacer páginas que parecen páginas web.

Además, esta unidad encaja de forma perfecta en la ruta del curso. Ya sabes maquetar el texto; ahora aprendes a agruparlo. Y más adelante, cuando llegues a CSS, vas a descubrir que casi todo el diseño visual de una página (colores, fondos, bordes, posiciones) se aplica precisamente sobre listas, tablas y divs. El CSS necesita elementos a los cuales agarrarse, y los divs y las listas son sus mejores amigos. Si esta unidad la dominas bien, el CSS te va a resultar mucho más fácil, porque ya vas a tener el hábito de pensar en bloques y grupos en lugar de pensar en texto suelto.

Otro punto importante: los elementos de esta unidad son los que más usan los desarrolladores profesionales todos los días. Abre el código fuente de cualquier sitio grande y vas a encontrar ul por todos lados (los menús de navegación casi siempre son listas), tablas en páginas de datos y reportes, y divs en cantidades industriales. No es un tema secundario: es el pan de cada día del desarrollo web. Por eso esta unidad es más larga y más práctica que las anteriores: necesitas ver muchos ejemplos, cometer los errores típicos en un ambiente seguro, y practicar hasta que te salgan de forma natural.

En la explicación detallada vamos a desmenuzar cada etiqueta con analogías que te ayuden a recordarla para siempre: las listas son como la lista del súper, las tablas son como una hoja de cálculo, los divs son como cajas de mudanza y los spans son como un resaltador de textos. También vas a aprender los errores más comunes que cometen los principiantes (como olvidar cerrar las etiquetas li, o el pecado mortal de usar tablas para maquetar el diseño de la página, o la enfermedad de la divitis que es usar divs para absolutamente todo) y las buenas prácticas que separan a un aficionado de un profesional.

Después vienen los ejemplos paso a paso, cuatro proyectos completos y comentados línea por línea: un menú de navegación con ul, una receta de cocina con ol y listas anidadas, una tabla de datos con caption, y una tarjeta de perfil construida con div y span. Cada ejemplo está pensado para que lo copies, lo pruebes en tu navegador y lo modifiques para hacerlo tuyo.

Luego la sección de práctica te propone cinco ejercicios con su solución explicada paso a paso, para que compruebes que de verdad lo entendiste. Y al final, un resumen con los puntos clave de la unidad para repasar rápido cuando lo necesites.

Así que ponte cómodo, abre tu editor de código y tu navegador, y prepárate: al terminar esta unidad vas a saber organizar cualquier tipo de información en una página web. Esa es una de las habilidades que más se notan cuando alguien ve tu trabajo. Vamos a darle.

Explicación detallada

Las listas: ul, ol y li

Qué es una lista en HTML

Una lista en HTML es exactamente lo que suena: un grupo de elementos relacionados que se presentan juntos. Piensa en la lista del súper que haces antes de ir al mercado: leche, huevos, tortillas, queso. Todos esos elementos pertenecen a la misma categoría (cosas que hay que comprar) y están al mismo nivel (ninguno es más importante que otro). En HTML, esa lista se representa con una etiqueta contenedora que dice "esto es una lista" y una etiqueta li por cada elemento que dice "esto es un item de la lista".

Hay dos tipos de listas, y la diferencia entre ellas es clave. La lista no ordenada (ul, de unordered list) presenta sus elementos con viñetas: puntitos, palomitas o el símbolo que el navegador ponga por defecto. Se usa cuando el orden no importa, como la lista del súper, los enlaces de un menú de navegación o las características de un producto. La lista ordenada (ol, de ordered list) presenta sus elementos con números: 1, 2, 3. Se usa cuando el orden sí importa, como los pasos de una receta, las instrucciones de un manual o un ranking de los mejores.

Fíjate en algo importante: el navegador se encarga de dibujar las viñetas o los números automáticamente. Tú solo escribes los items con li y el navegador hace el resto. Esto es una muestra del poder del HTML semántico: tú describes qué significa el contenido (es una lista) y el navegador decide cómo presentarlo.

La etiqueta ul: lista no ordenada

La sintaxis básica de una lista no ordenada es así:

```
<ul>
  <li>Leche</li>
  <li>Huevos</li>
  <li>Tortillas</li>
  <li>Queso</li>
</ul>
```

El navegador muestra esto con una viñeta delante de cada item: un puntito redondo por defecto. La etiqueta ul es el contenedor y cada li es un elemento. Nunca debes poner un li suelto fuera de un ul o un ol: sería como escribir un renglón de la lista del súper en una hoja separada, pierde todo su sentido.

¿Para qué se usa ul en la vida real? El uso más famoso es el menú de navegación. Casi todos los menús de los sitios web están hechos con una lista no ordenada: cada opción del menú es un li que contiene un enlace. Tiene lógica: las opciones de un menú son un grupo de elementos relacionados al mismo nivel, y el orden normalmente no es crítico. Otros usos típicos: lista de características de un producto, lista de servicios de una empresa, lista de enlaces relacionados, etiquetas de un artículo, miembros de un equipo.

Ejemplo de menú de navegación:

```
<nav>
  <ul>
    <li><a href="index.html">Inicio</a></li>
    <li><a href="cursos.html">Cursos</a></li>
    <li><a href="blog.html">Blog</a></li>
    <li><a href="contacto.html">Contacto</a></li>
  </ul>
</nav>
```

Aquí ves algo muy común: etiquetas anidadas. Dentro de cada li hay un enlace a. Esto es perfectamente válido y es el patrón estándar de los menús. Más adelante, con CSS, vas a quitar las viñetas y poner estas opciones en horizontal, pero la base HTML siempre es esta lista.

La etiqueta ol: lista ordenada

La sintaxis de la lista ordenada es idéntica, solo cambia el contenedor:

```
<ol>
  <li>Hervir el agua</li>
  <li>Agregar la pasta</li>
  <li>Cocinar ocho minutos</li>
  <li>Escurrir y servir</li>
</ol>
```

El navegador numera automáticamente: 1, 2, 3, 4. Si agregas un paso en medio, no tienes que reenumerar nada: el navegador recalcula los números solo. Esta es una ventaja enorme frente a escribir los números a mano. Imagina una receta de 20 pasos donde decides agregar un paso nuevo entre el 5 y el 6: si los números estuvieran escritos a mano tendrías que corregir los 15 restantes; con ol solo insertas el li y listo.

La etiqueta `ol` tiene tres atributos muy útiles que vale la pena conocer a fondo: `type`, `start` y `reversed`.

El atributo `type` cambia el estilo de la numeración. Los valores posibles son: 1 para números arábigos (es lo normal, 1, 2, 3), `a` para letras minúsculas (a, b, c), `A` para letras mayúsculas (A, B, C), `i` para números romanos en minúsculas (i, ii, iii) y `I` para números romanos en mayúsculas (I, II, III). Ejemplo:

```
<ol type="A">
  <li>Primera opción</li>
  <li>Segunda opción</li>
  <li>Tercera opción</li>
</ol>
```

Esto se muestra como A, B, C. Es perfecto para exámenes, cuestionarios o documentos formales donde las opciones se identifican con letras.

El atributo `start` indica desde qué número empieza la cuenta. Por defecto empieza en 1, pero puedes cambiarlo:

```
<ol start="5">
  <li>Quinto paso</li>
  <li>Sexto paso</li>
</ol>
```

Esto es útil cuando una lista continúa a otra lista anterior, por ejemplo cuando un procedimiento largo se divide en varias secciones y quieres que la numeración no se reinicie. También se usa en combinación con `type`: si `type` es `A` y `start` es 3, la lista empieza en C.

El atributo `reversed` invierte el orden de la numeración: la lista cuenta hacia atrás. Es un atributo booleano, lo que significa que no necesita valor, solo escribirlo lo activa:

```
<ol reversed>
  <li>Tercer lugar</li>
  <li>Segundo lugar</li>
  <li>Primer lugar</li>
</ol>
```

Esto se muestra como 3, 2, 1. Es ideal para conteos regresivos, rankings donde quieres mostrar del último al primero, o listas de "top 10" donde el número 1 queda al final como gran revelación. Puedes combinarlo con start: una lista con start 10 y reversed cuenta 10, 9, 8 hacia abajo.

Un detalle importante: aunque type, start y reversed controlan la apariencia de los números, hoy en día muchos desarrolladores prefieren controlar la numeración con CSS en lugar de estos atributos, porque CSS ofrece más opciones visuales. Sin embargo, los atributos siguen siendo válidos en HTML5 y son la forma más rápida de lograr estos efectos sin escribir ni una línea de CSS. Conócelos, úsalos cuando te convengan, y cuando aprendas CSS tendrás la alternativa más potente.

La etiqueta li: el elemento de lista

La etiqueta li (de list item) representa cada elemento individual dentro de una lista. Es una etiqueta sencilla pero con reglas claras: siempre va dentro de un ul, un ol o un menú, y siempre debe cerrarse. Su contenido puede ser texto simple o cualquier otro elemento HTML: enlaces, imágenes, párrafos, incluso otras listas completas (eso son las listas anidadas, que veremos enseguida).

Algo que muchos principiantes no saben: un li puede contener varias líneas de contenido, no solo una frase corta. Puedes meter un párrafo completo, una imagen con su descripción, o un bloque entero dentro de un li. Por ejemplo, en una lista de noticias, cada li podría contener un titular, una imagen y un resumen. La lista solo dice "estos son elementos hermanos del mismo grupo"; lo que hay dentro de cada uno puede ser tan complejo como quieras.

Listas anidadas: listas dentro de listas

Una de las técnicas más poderosas de las listas es anidarlas: poner una lista completa dentro de un li de otra lista. Esto sirve para representar jerarquías, como un índice de un libro donde cada capítulo tiene sus secciones, o un menú con submenús.

La regla de oro de las listas anidadas es: la lista hija siempre va DENTRO del li del padre, no después de él. Mira la diferencia:

Forma correcta:


```
<ul>
  <li>Frutas
    <ul>
      <li>Manzana</li>
      <li>Plátano</li>
      <li>Naranja</li>
    </ul>
  </li>
  <li>Verduras
    <ul>
      <li>Zanahoria</li>
      <li>Lechuga</li>
    </ul>
  </li>
</ul>
```

Fíjate con cuidado: el ul anidado está dentro del li de "Frutas", antes de que se cierre ese li. El navegador entiende entonces que Manzana, Plátano y Naranja son hijos de Frutas, y los muestra con una sangría (un espacio hacia adentro) y normalmente con un estilo de viñeta distinto para que se note el nivel.

La forma incorrecta, que muchos principiantes escriben, es cerrar el li y luego poner la lista anidada como hermana:

```
<ul>
  <li>Frutas</li>
  <ul>
    <li>Manzana</li>
  </ul>
</ul>
```

Esto funciona visualmente en la mayoría de los navegadores, pero es HTML inválido: un ul no puede ser hijo directo de otro ul, solo los li pueden ser hijos de una lista. Los validadores de HTML lo marcan como error, y los lectores de pantalla (las herramientas que usan las personas con discapacidad visual) pueden confundirse. Hazlo bien desde el principio: lista hija siempre dentro del li.

Las listas anidadas también pueden mezclar tipos: un ol dentro de un ul, o un ul dentro de un ol. Por ejemplo, una receta donde cada paso (ol) tiene una lista de ingredientes o notas (ul):

```
<ol>
  <li>Prepara la masa
    <ul>
      <li>500 gramos de harina</li>
      <li>250 mililitros de agua tibia</li>
      <li>Una pizca de sal</li>
    </ul>
  </li>
  <li>Deja reposar 30 minutos</li>
  <li>Hornea a 200 grados</li>
</ol>
```

Este patrón de pasos ordenados con detalles no ordenados es extremadamente común en tutoriales, manuales y recetas. Domínalo y lo vas a usar toda tu carrera.

Usos profesionales de las listas

Vale la pena detenerse en los usos reales porque aquí es donde las listas dejan de ser un ejercicio de clase y se convierten en herramienta de trabajo. Los menús de navegación, como ya vimos, son el uso número uno: prácticamente todos los sitios usan ul para sus menús. Las migas de pan (esos enlaces tipo Inicio > Cursos > HTML5 que ves arriba de muchas páginas) también son listas. Los índices o tablas de contenido de artículos largos son listas, a veces anidadas. Las listas de preguntas frecuentes, los pasos de un tutorial, los ingredientes de una receta, las características de un producto, los testimonios de clientes, los miembros de un equipo, las redes sociales en el pie de página: todo eso son listas en el HTML real.

Hay una razón profunda por la que los profesionales aman las listas para estas cosas: son semánticas. Cuando un lector de pantalla encuentra un ul, le dice al usuario "lista con 5 elementos", y el usuario puede saltarla o recorrerla elemento por elemento. Si en cambio hubieras puesto cinco divs sueltos, el lector de pantalla no sabría que están relacionados. La accesibilidad no es un lujo: es parte de hacer bien tu trabajo, y las listas son una de las formas más fáciles de lograrla.

Errores comunes con listas

El error número uno es olvidar cerrar las etiquetas li. Técnicamente, en HTML5 el cierre de li es opcional en algunos casos (el navegador lo infiere), pero es una pésima práctica omitirlo: el código se vuelve difícil de leer, los errores se multiplican cuando anidas listas, y otros desarrolladores van a sufrir al leer tu código. Cierra siempre tus li. Sin excepciones.

El error número dos es poner contenido directamente dentro de `ul` u `ol` sin envolverlo en `li`. Por ejemplo, escribir texto suelto o un enlace directamente dentro del `ul`. Solo los `li` pueden ser hijos directos de una lista. Si necesitas un encabezado antes de la lista, ponlo fuera del `ul`, no adentro.

El error número tres es usar listas para cosas que no son listas. No todo lo que se ve en vertical es una lista. Si tienes tres párrafos de texto corrido, no los conviertas en `li` solo porque quieres viñetas: usa párrafos y ponles estilo con CSS. La lista debe representar una relación real de grupo entre sus elementos.

El error número cuatro es anidar mal las listas, poniendo la lista hija fuera del `li` como vimos antes. Repito la regla porque es la que más se viola: la lista anidada va dentro del `li`, antes de su cierre.

Las tablas: `table`, `tr`, `th`, `td` y `caption`

Qué es una tabla en HTML

Una tabla en HTML es una cuadrícula para presentar datos relacionados en filas y columnas. La analogía perfecta es una hoja de cálculo: piensa en Excel o en Google Sheets. Tienes columnas con un significado (nombre, edad, ciudad) y filas con los registros (cada persona). La tabla HTML funciona exactamente igual: es la herramienta correcta cuando tus datos tienen esa estructura de dos dimensiones.

Es importante entender desde ahora la regla de oro de las tablas: las tablas son para datos tabulares, no para maquetar el diseño de la página. Hace muchos años, cuando CSS todavía no existía o era muy limitado, los desarrolladores usaban tablas para acomodar el diseño completo de sus páginas: el encabezado en una celda, el menú en otra, el contenido en otra. Eso era un truco sucio de la época y hoy es considerado una mala práctica grave. Las tablas para layout producen código horrible, páginas lentas, pesadillas de mantenimiento y desastres de accesibilidad. Para el diseño visual existe CSS, y lo aprenderás más adelante. Las tablas se quedan con su trabajo original y legítimo: mostrar datos que de verdad son tabulares, como horarios, precios, estadísticas, comparativas y reportes.

La estructura básica: `table`, `tr`, `th`, `td`

Una tabla se construye con cuatro etiquetas principales que trabajan en equipo. La etiqueta `table` es el contenedor de toda la tabla. Dentro de `table` van las filas, cada una marcada con `tr` (de `table row`, fila de tabla). Y dentro de cada fila van las celdas: `th` (de `table header`,

encabezado de tabla) para las celdas de encabezado, y td (de table data, dato de tabla) para las celdas de datos normales.

Vamos a construir la tabla del ejemplo clásico, nombre y edad, paso por paso para que veas cómo piensa una tabla:

```
<table>
  <tr>
    <th>Nombre</th>
    <th>Edad</th>
  </tr>
  <tr>
    <td>Ana</td>
    <td>25</td>
  </tr>
  <tr>
    <td>Luis</td>
    <td>30</td>
  </tr>
  <tr>
    <td>Marta</td>
    <td>22</td>
  </tr>
</table>
```

Analicemos la lógica. La tabla tiene tres filas (tr): la primera es la fila de encabezados, con dos th que dicen "Nombre" y "Edad". Las siguientes dos filas son datos, cada una con dos td. Cada fila tiene exactamente el mismo número de celdas: dos. Esa regularidad es lo que hace que la tabla se vea como una cuadrícula ordenada. Si una fila tuviera tres celdas y las demás dos, la tabla se descuadraría y se vería rota. La consistencia en el número de celdas por fila es una regla práctica fundamental.

Fíjate también en la diferencia visual entre th y td: por defecto, los navegadores muestran el contenido de th en negritas y centrado, mientras que el td se muestra normal y alineado a la izquierda. Esa diferencia no es decorativa: le dice al lector "esta fila o columna es el encabezado, aquí está la descripción de los datos". Y para los lectores de pantalla es aún más importante: un lector de pantalla usa los th para anunciar "Nombre: Ana, Edad: 25", relacionando cada dato con su encabezado. Si usaras td para todo, esa relación se perdería.

Una pregunta que surge de inmediato: ¿los th siempre van en la primera fila? No necesariamente. Los th pueden ir en la primera fila (encabezados de columna, lo más común)

o en la primera columna de cada fila (encabezados de fila). Por ejemplo, en una tabla de horarios, la primera columna podría tener los días de la semana como th, y las demás columnas los horarios como td:

```
<table>
  <tr>
    <th>Día</th>
    <th>Apertura</th>
    <th>Cierre</th>
  </tr>
  <tr>
    <th>Lunes</th>
    <td>9:00</td>
    <td>18:00</td>
  </tr>
  <tr>
    <th>Martes</th>
    <td>9:00</td>
    <td>18:00</td>
  </tr>
</table>
```

Aquí "Lunes" y "Martes" son th porque son encabezados de sus filas: describen qué significan los datos de esa fila. Esta flexibilidad hace que th sea más que "la fila de arriba": es "la celda que describe a las demás".

La etiqueta caption: el título de la tabla

Toda tabla que se respete debería tener un caption: es su título descriptivo. Se escribe como primer hijo de table, antes de cualquier tr, y el navegador lo muestra centrado arriba de la tabla por defecto:

```
<table>
  <caption>Edades de los estudiantes del grupo A</caption>
  <tr>
    <th>Nombre</th>
    <th>Edad</th>
  </tr>
  <tr>
    <td>Ana</td>
    <td>25</td>
  </tr>
</table>
```

¿Por qué importa el caption? Por dos razones. Primero, accesibilidad: los lectores de pantalla anuncian el caption antes de leer la tabla, así el usuario sabe de qué trata antes de escuchar los datos. Segundo, claridad: una tabla sin título obliga al lector a adivinar el contexto mirando los datos; con caption, el contexto está servido. Es una etiqueta de una línea que mejora muchísimo tu tabla. Úsala siempre.

El atributo border como concepto histórico

Si buscas tutoriales viejos de HTML, vas a encontrar el atributo border en las tablas: `<table border="1">`. Este atributo dibujaba un borde alrededor de la tabla y sus celdas. En HTML5, border como atributo presentacional está obsoleto: la forma correcta de poner bordes hoy es con CSS. Lo menciono por tres razones: para que no te asustes si lo ves en código antiguo, para que entiendas por qué tus tablas se ven "sin líneas" por defecto (el navegador no dibuja bordes a menos que se lo pidas con CSS), y para que no lo uses en código nuevo. Cuando llegues a CSS, aprenderás a darles a tus tablas bordes bonitos, colores de fondo por fila y todos los estilos que quieras. Por ahora, acepta que tu tabla se ve simple: lo importante es que la estructura sea correcta.

Tablas más grandes: pensando en filas y columnas

Vamos a subir un nivel con una tabla más realista, como la lista de precios de un servicio o el horario de clases de la academia:

```

<table>
  <caption>Horario de clases en vivo</caption>
  <tr>
    <th>Curso</th>
    <th>Día</th>
    <th>Hora</th>
    <th>Instructor</th>
  </tr>
  <tr>
    <td>HTML5 desde cero</td>
    <td>Lunes</td>
    <td>18:00</td>
    <td>Esteban</td>
  </tr>
  <tr>
    <td>CSS moderno</td>
    <td>Miércoles</td>
    <td>18:00</td>
    <td>Esteban</td>
  </tr>
  <tr>
    <td>JavaScript básico</td>
    <td>Viernes</td>
    <td>18:00</td>
    <td>Esteban</td>
  </tr>
</table>

```

Cuatro columnas, cuatro filas (una de encabezados y tres de datos). Cada fila tiene exactamente cuatro celdas. El caption explica de qué trata. Los th describen cada columna. Esto es una tabla profesional en su estructura, aunque todavía sin estilos. Cuando le agregues CSS más adelante, esta misma estructura se va a ver espectacular sin que tengas que tocar el HTML.

Un consejo práctico para tablas grandes: escríbelas fila por fila y cuenta las celdas de cada fila mientras escribes. Es muy fácil equivocarse y poner una celda de más o de menos, y el descuadre resultante es confuso. Algunos editores de código te ayudan con el autocompletado, pero el hábito de contar te va a salvar muchas veces.

Errores comunes con tablas

El error más grave, como ya dijimos, es usar tablas para maquetar el diseño de la página. Si te descubres metiendo el menú, el contenido y el pie de página en celdas de una tabla gigante, detente: ese es el camino equivocado. Para layout existen los divs y CSS. Las tablas son solo para datos tabulares.

El segundo error es olvidar cerrar las etiquetas `tr`, `th` o `td`. Como las tablas tienen muchas etiquetas anidadas, es fácil perder la cuenta. Un `tr` sin cerrar puede hacer que filas enteras se fusionen de formas extrañas. Escribe con cuidado y revisa.

El tercer error es mezclar el número de celdas por fila sin querer. Cada fila debe tener el mismo número de celdas para que la cuadrícula quede pareja. (Existen los atributos `colspan` y `rowspan` para fusionar celdas a propósito, pero eso es tema de una unidad más avanzada; por ahora, mantén la regularidad.)

El cuarto error es no usar `th` para los encabezados y poner todo con `td`, o peor, simular encabezados con negritas dentro de `td`. Los `th` no son decoración: son información semántica que ayuda a los lectores de pantalla y a los buscadores a entender tu tabla. Úsalos.

El quinto error es olvidar el `caption`. Ya sabes: una línea, mucho beneficio. No la olvides.

Los contenedores: `div` y `span`

Qué son los contenedores genéricos

Llegamos a las dos etiquetas más usadas de todo el HTML: `div` y `span`. Son contenedores genéricos, lo que significa que no tienen un significado semántico propio como `p` (párrafo) o `h1` (encabezado principal). Su trabajo es agrupar contenido para que puedas tratarlo como una unidad: darle estilo con CSS, manipularlo con JavaScript, o simplemente organizar tu documento en bloques lógicos.

La diferencia fundamental entre ellos es el tipo de caja que crean. El `div` es un elemento de bloque: ocupa todo el ancho disponible y siempre empieza en una línea nueva, como si fuera un párrafo gigante sin formato. El `span` es un elemento en línea: ocupa solo el espacio de su contenido y no rompe la línea, como si fuera una palabra más dentro del texto. Esta diferencia entre bloque y en línea es uno de los conceptos más importantes de todo el desarrollo web, así que vamos a desmenuzarla a fondo.

La analogía de las cajas de mudanza y el resaltador

Piensa en el div como una caja de mudanza. Cuando te mudas, no llevas tus cosas sueltas una por una: las metes en cajas, y cada caja agrupa cosas relacionadas (caja de libros, caja de trastes de cocina, caja de ropa). La caja no cambia lo que hay adentro, solo lo agrupa para que puedas moverlo, etiquetarlo y tratarlo como una unidad. El div hace exactamente eso con tu contenido HTML: agrupa un conjunto de elementos (un título, un párrafo, una imagen, un botón) en una caja lógica que luego puedes estilizar o mover como un todo.

Piensa en el span como un resaltador de textos. Cuando estudias y quieres destacar una frase importante dentro de un párrafo, pasas el resaltador amarillo sobre esas palabras. El resaltador no rompe el párrafo ni mueve el texto a otro lado: solo marca un fragmento dentro del flujo del texto. El span hace eso: marca un pedacito de texto dentro de un elemento mayor para que puedas darle un estilo distinto (otro color, negritas, fondo) sin alterar la estructura.

Con estas dos analogías ya tienes la intuición correcta: div agrupa bloques grandes, span marca fragmentos pequeños dentro del texto.

El div: bloque contenedor

La sintaxis del div es simple:

```
<div>
  <h2>Mi sección</h2>
  <p>Este es un párrafo dentro del div.</p>
  <p>Este es otro párrafo dentro del mismo div.</p>
</div>
```

El div envuelve a sus hijos y los trata como un grupo. Por defecto, visualmente no pasa nada espectacular: el div no tiene bordes, ni fondo, ni márgenes especiales. Es invisible hasta que le aplicas CSS. Y esa es precisamente su gracia: es un contenedor neutro esperando a que tú le des propósito con estilos.

¿Dónde se usa el div? En todas partes. Cada sección de una página suele ser un div: el encabezado del sitio, la barra de navegación, el contenido principal, la barra lateral, el pie de página. Cada tarjeta de un catálogo es un div. Cada bloque de un formulario es un div. Cuando veas el código de un sitio profesional, vas a ver divs anidados dentro de divs formando la estructura completa de la página. Son los ladrillos de la construcción web.

Ejemplo de tarjeta de producto con div:

```
<div>
  <h3>Laptop ultradelgada</h3>
  <p>Procesador de última generación, 16 GB de memoria.</p>
  <p>Precio: $15,999</p>
  <a href="comprar.html">Comprar ahora</a>
</div>
```

Todo lo relacionado con ese producto está dentro de un solo div. Más adelante, con CSS, le darás a ese div un borde, un fondo, una sombra y un ancho fijo, y se verá como una tarjeta. Pero la agrupación lógica ya está hecha aquí, en el HTML, con el div.

El span: texto en línea

El span se usa dentro de otros elementos para marcar fragmentos de texto:

```
<p>El precio normal es $1,000 pero <span>hoy cuesta $799</span> por oferta.</p>
```

El span no rompe el párrafo: el texto sigue fluyendo en la misma línea. Por defecto tampoco se ve distinto: igual que el div, espera a que CSS le dé estilo. Su poder está en que te permite aplicar estilo a una parte del texto sin afectar al resto. Con CSS podrías hacer que ese span se vea en rojo y en negritas, destacando la oferta dentro del párrafo.

Otros usos típicos del span: resaltar una palabra clave dentro de un título, cambiar el color de una parte de un texto, marcar un dato (como un precio o una fecha) para que JavaScript lo encuentre y lo manipule, o aplicar un estilo distinto a la primera letra de un párrafo.

Bloque contra en línea: la diferencia que lo explica todo

Vamos a profundizar en la diferencia entre elementos de bloque y elementos en línea, porque es la llave que abre la comprensión del diseño web.

Los elementos de bloque (como div, p, h1 a h6, ul, ol, li, table) comparten tres comportamientos: ocupan todo el ancho disponible de su contenedor, empiezan siempre en una línea nueva (empujan lo que sigue hacia abajo), y pueden contener otros elementos de bloque y en línea. Piensa en ellos como cajas que se apilan verticalmente una sobre otra, como los pisos de un edificio.

Los elementos en línea (como span, a, strong, em, img) comparten los comportamientos opuestos: ocupan solo el ancho de su contenido, no rompen la línea (fluyen junto al texto que los rodea), y normalmente solo contienen texto u otros elementos en línea. Piensa en ellos

como palabras dentro de un renglón: se acomodan una tras otra hasta llenar la línea y luego continúan en la siguiente.

Mira este ejemplo que muestra la diferencia en acción:

```
<div>Este div ocupa todo el ancho y empuja lo siguiente hacia abajo.</div>  
<div>Este es otro div: aparece en una línea nueva.</div>  
<p>Este párrafo tiene <span>un span adentro</span> que no rompe la línea.</p>
```

Los dos divs se muestran uno debajo del otro, cada uno en su propia línea, aunque su texto sea corto. El span, en cambio, vive feliz dentro del párrafo sin alterar el flujo del texto. Esa es toda la diferencia, y sin embargo de ella dependen cosas enormes: por qué los menús de navegación necesitan CSS para ponerse horizontales (porque los li son de bloque y se apilan), por qué puedes poner varios enlaces seguidos en un párrafo sin que se rompan (porque los a son en línea), y por qué los divs son la base del layout (porque su comportamiento de bloque los hace predecibles para apilar secciones).

Una regla práctica de anidamiento que debes respetar: los elementos de bloque pueden contener elementos en línea y otros bloques, pero los elementos en línea no deben contener elementos de bloque. Es decir, puedes meter un span dentro de un div (correcto), pero no debes meter un div dentro de un span o dentro de un párrafo (incorrecto). Tiene lógica: no puedes meter una caja de mudanza dentro de una palabra resaltada. Los navegadores a veces lo "perdonan" y lo muestran de todos modos, pero es HTML inválido y te puede dar sorpresas desagradables con CSS.

Cuándo usar div y cuándo usar span

La regla es sencilla: usa div cuando necesites agrupar bloques de contenido (secciones, tarjetas, columnas, áreas de la página) y usa span cuando necesites marcar un fragmento de texto dentro de una línea. Si lo que agrupas va a ocupar su propio espacio vertical en la página, es div. Si es un pedacito dentro de una frase, es span.

Pero aquí viene una advertencia importante de buenas prácticas: no uses div para todo. Existe una enfermedad muy común entre principiantes llamada divitis: la inflamación crónica de divs, que consiste en envolver absolutamente todo en divs incluso cuando existen etiquetas semánticas más apropiadas. Si tienes un párrafo, usa p, no div. Si tienes un encabezado, usa h1, no div. Si tienes una lista, usa ul, no divs con viñetas falsas. El div es para agrupar, no para sustituir a las demás etiquetas. Un HTML lleno de divs sin sentido es difícil de leer, malo para el posicionamiento en buscadores y problemático para la accesibilidad. Usa div con propósito, no por costumbre.

Lo mismo vale para span: no envuelvas palabras en span "por si acaso". El span debe tener un propósito: un estilo que vas a aplicar, un dato que JavaScript va a manipular, una parte del texto que necesita tratamiento distinto. Spans vacíos o sin propósito solo ensucian el código.

Div y span en el mundo real

Para que veas el poder real de estos contenedores, imagina la página principal de la academia. Su estructura con divs sería algo así:

```
<div>
  <h1>Lucas EGA Academy</h1>
  <p>Aprende tecnología en español, gratis.</p>
</div>
<div>
  <h2>Cursos populares</h2>
  <div>
    <h3>HTML5 desde cero</h3>
    <p>30 clases, 100 por ciento gratis.</p>
  </div>
  <div>
    <h3>CSS moderno</h3>
    <p>30 clases, 100 por ciento gratis.</p>
  </div>
  <div>
    <h3>JavaScript básico</h3>
    <p>30 clases, 100 por ciento gratis.</p>
  </div>
</div>
<div>
  <p>Contacto: <span>hola@lucasegaacademy.com</span></p>
</div>
```

Tres grandes bloques (divs): el encabezado, la sección de cursos y el pie. Dentro de la sección de cursos, tres tarjetas (divs), una por curso. Y dentro del pie, un span marcando el correo para darle un estilo distinto. Esta es la anatomía de una página real, y está construida casi por completo con divs y spans. Cuando le agregues CSS, cada div se convertirá en una zona visual con su propio diseño, y el span del correo se verá destacado. El HTML organiza, el CSS decora: esa es la división del trabajo.

Errores comunes con div y span

El primer error es la divitis, que ya mencionamos: usar divs para todo en lugar de las etiquetas semánticas correctas. Cada vez que escribas un div, pregúntate: ¿esto es realmente un grupo genérico, o debería ser un p, un ul, un h2? Si hay una etiqueta más específica, úsala.

El segundo error es olvidar cerrar los divs, especialmente cuando están anidados. Un div sin cerrar puede tragarse secciones enteras de tu página y el resultado visual es un desastre difícil de diagnosticar. La indentación (el sangrado del código) es tu mejor aliada: cada nivel de anidamiento se sangra un poco más, y así ves de un vistazo qué cierra a qué. Todos los ejemplos de este libro usan indentación consistente por esa razón: cópiala como hábito.

El tercer error es meter bloques dentro de elementos en línea, como un div dentro de un span o dentro de un p. Ya sabes la regla: los en línea no contienen bloques. Si necesitas agrupar bloques, el contenedor debe ser un div (u otro bloque), nunca un span.

El cuarto error es usar span como si fuera div, esperando que se comporte como bloque. Si pones un span y esperas que ocupe todo el ancho o que empuje el contenido hacia abajo, te vas a frustrar: el span es en línea por naturaleza. Para agrupar bloques usa div.

El quinto error es abusar de divs y spans vacíos como "espaciadores": divs sin contenido cuya única función es crear espacio en blanco en la página. El espacio visual es trabajo de CSS (márgenes y rellenos), no de etiquetas vacías en el HTML. Un HTML limpio no tiene elementos vacíos decorativos.

Buenas prácticas de la unidad

Para cerrar la explicación, reunamos las buenas prácticas que aplican a los tres temas. Primero, piensa en el significado antes que en la apariencia: elige ul u ol según si el orden importa, usa th para encabezados de verdad, y usa div y span para agrupar con propósito. El HTML describe qué es el contenido; de cómo se ve se encarga CSS.

Segundo, mantén la estructura válida: li siempre dentro de ul u ol, listas anidadas dentro del li, el mismo número de celdas por fila en las tablas, caption como primer hijo de table, y bloques conteniendo en líneas pero no al revés. Un HTML válido es un HTML predecible: se ve igual en todos los navegadores y funciona bien con lectores de pantalla.

Tercero, usa la indentación de forma religiosa. Con listas anidadas, tablas de muchas filas y divs dentro de divs, la única forma de no perderte es sangrar cada nivel. Tu yo del futuro (y cualquier persona que lea tu código) te lo va a agradecer.

Cuarto, no tengas miedo de combinar estos elementos. Las herramientas más poderosas surgen de combinarlas: una tarjeta de producto (div) que contiene una lista de características (ul), una tabla de precios dentro de una sección (div), un menú de navegación (ul) dentro del encabezado (div). El HTML profesional es una orquesta de estos elementos trabajando juntos, no una sola etiqueta haciendo todo.

Y quinto, practica con contenido real. Los ejemplos abstractos (elemento 1, elemento 2) sirven para entender la sintaxis, pero el aprendizaje de verdad ocurre cuando construyes cosas que te importan: el menú de tu propio sitio, la tabla de tus gastos, la tarjeta de tu perfil. Los ejercicios de esta unidad están diseñados justo para eso.

Anatomía de una página real: todo junto

Ya viste cada herramienta por separado. Ahora veamos cómo trabajan juntas en una página de verdad. Imagina la página de un curso de la academia. Su esqueleto con los elementos de esta unidad se ve así:

```

<div>
  <h1>Curso de HTML5 desde cero</h1>
  <p>El curso <span>más popular</span> de la academia, con 30 clases gratis.</p>
  <ul>
    <li><a href="#temario">Temario</a></li>
    <li><a href="#horarios">Horarios</a></li>
    <li><a href="#inscripcion">Inscripción</a></li>
  </ul>
</div>
<div>
  <h2 id="temario">Temario</h2>
  <ol>
    <li>Qué es HTML y cómo funciona</li>
    <li>Estructura de un documento</li>
    <li>Listas, tablas y contenedores</li>
    <li>Formularios</li>
  </ol>
  <h2 id="horarios">Horarios</h2>
  <table>
    <caption>Grupos disponibles</caption>
    <tr>
      <th>Grupo</th>
      <th>Días</th>
      <th>Hora</th>
    </tr>
    <tr>
      <td>Matutino</td>
      <td>Lunes y miércoles</td>
      <td>9:00</td>
    </tr>
    <tr>
      <td>Vespertino</td>
      <td>Martes y jueves</td>
      <td>18:00</td>
    </tr>
  </table>
</div>
<div>
  <p>Costo: <span>$0</span>. Incluye <span>certificado</span> al terminar.</p>
</div>

```

Tres divs grandes: encabezado, contenido y pie. Dentro del encabezado hay un menú hecho con ul. Dentro del contenido hay un temario hecho con ol y una tabla de horarios con caption.

Los spans destacan "más popular", "\$0" y "certificado". Cada elemento hace exactamente el trabajo para el que fue diseñado, y ninguno invade el terreno de otro. Cuando veas una página profesional y te preguntes "¿cómo está construida?", la respuesta casi siempre es una variación de este patrón: divs que agrupan, listas que organizan, tablas que tabulan y spans que destacan.

Fíjate además en un detalle: los enlaces del menú apuntan a #temario, #horarios e #inscripción. Esos son enlaces internos que saltan a secciones de la misma página (los id correspondientes irían en los encabezados). Combinar listas de navegación con anclas internas es una técnica clásica de páginas largas: el menú de arriba te lleva a cada sección. No necesitas JavaScript ni nada avanzado: puro HTML de esta unidad.

Tres casos guiados

Para que el aprendizaje quede bien amarrado, resolvamos tres casos típicos como los que te vas a encontrar en proyectos reales. No son ejercicios para entregar, son lecturas guiadas: sigue el razonamiento y luego intenta replicarlos por tu cuenta.

Caso 1: el menú de un restaurante. Un restaurante necesita mostrar sus platillos organizados por categorías: entradas, platos fuertes, postres y bebidas. Cada categoría tiene varios platillos, y algunos platillos tienen una nota (picante, vegetariano, de la casa). La herramienta natural es una lista no ordenada de categorías, con una lista anidada de platillos dentro de cada una, y spans para las notas destacadas:


```

<h2>Menú del día</h2>
<ul>
  <li>Entradas
    <ul>
      <li>Guacamole con totopos <span>(de la casa)</span></li>
      <li>Queso fundido</li>
      <li>Sopa de tortilla</li>
    </ul>
  </li>
  <li>Platos fuertes
    <ul>
      <li>Enchiladas verdes <span>(picante)</span></li>
      <li>Pozole rojo</li>
      <li>Ensalada César <span>(vegetariano)</span></li>
    </ul>
  </li>
  <li>Postres
    <ul>
      <li>Flan napolitano</li>
      <li>Arroz con leche</li>
    </ul>
  </li>
</ul>

```

El razonamiento: las categorías son un grupo sin orden importante, así que `ul`. Los platillos dentro de cada categoría también son un grupo sin orden, así que `ul` anidado dentro del `li` de la categoría. Las notas como "(picante)" son fragmentos dentro de la línea del platillo, así que `span`: con CSS luego les das color rojo o una etiquetita visual. Todo encaja.

Caso 2: la tabla de posiciones de un torneo. Un torneo de fútbol necesita mostrar equipos, partidos jugados, ganados, empatados, perdidos y puntos. Esto grita "tabla": los datos son tabulares por naturaleza, con columnas que describen y filas que registran.

```

<table>
  <caption>Tabla de posiciones, torneo de barrio, jornada 8</caption>
  <tr>
    <th>Pos</th>
    <th>Equipo</th>
    <th>PJ</th>
    <th>G</th>
    <th>E</th>
    <th>P</th>
    <th>Pts</th>
  </tr>
  <tr>
    <td>1</td>
    <td>Real Colonia</td>
    <td>8</td>
    <td>6</td>
    <td>1</td>
    <td>1</td>
    <td>19</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Atlético Mercado</td>
    <td>8</td>
    <td>5</td>
    <td>2</td>
    <td>1</td>
    <td>17</td>
  </tr>
  <tr>
    <td>3</td>
    <td>Deportivo Tacos</td>
    <td>8</td>
    <td>4</td>
    <td>2</td>
    <td>2</td>
    <td>14</td>
  </tr>
</table>

```

El razonamiento: siete columnas bien definidas con th (Pos, Equipo, PJ, G, E, P, Pts), tres filas de datos con td, caption que ubica el contexto (qué torneo, qué jornada). Cada fila tiene exactamente siete celdas. Si mañana se juega la jornada 9, agregas o actualizas filas sin tocar

la estructura. Esta tabla, con un poco de CSS (encabezado con fondo verde, filas alternadas), se vería idéntica a las de los sitios deportivos profesionales.

Caso 3: la página de aterrizaje de un curso. Una landing sencilla necesita: un encabezado con el nombre y un menú, una sección de beneficios, una tabla comparativa de planes y un pie con contacto. Aquí se luce la combinación completa:

```

<div>
  <h1>Academia de inglés en línea</h1>
  <ul>
    <li><a href="#beneficios">Beneficios</a></li>
    <li><a href="#planes">Planes</a></li>
    <li><a href="#contacto">Contacto</a></li>
  </ul>
</div>
<div>
  <h2 id="beneficios">Beneficios</h2>
  <ul>
    <li>Clases <span>en vivo</span> todos los días</li>
    <li>Grupos de máximo <span>10 personas</span></li>
    <li>Certificado al terminar cada nivel</li>
  </ul>
  <h2 id="planes">Planes</h2>
  <table>
    <caption>Comparativa de planes</caption>
    <tr>
      <th>Plan</th>
      <th>Precio mensual</th>
      <th>Clases por semana</th>
    </tr>
    <tr>
      <td>Básico</td>
      <td>$299</td>
      <td>2</td>
    </tr>
    <tr>
      <td>Avanzado</td>
      <td>$499</td>
      <td>5</td>
    </tr>
  </table>
</div>
<div>
  <p id="contacto">Escríbenos a <span>info@academiaingles.com</span> o llámanos al <span>55 1234 5678</span>.</p>
</div>

```

El razonamiento: tres divs para las tres zonas de la página. Menú con ul y anclas internas. Beneficios con ul porque son un grupo sin orden, con spans para destacar "en vivo" y "10 personas". Comparativa de planes con table porque son datos tabulares: columnas Plan,

Precio y Clases, dos filas de datos. Contacto con spans para el correo y el teléfono, que con CSS se verán destacados o se convertirán en enlaces. Esta landing ya es un proyecto digno de un portafolio de principiante, y está construida al cien por ciento con lo que aprendiste en esta unidad.

Preguntas frecuentes de la unidad

¿Puedo poner una lista dentro de una tabla? Sí. Una celda td o th puede contener casi cualquier elemento, incluyendo listas completas. Por ejemplo, una celda de "ingredientes" podría contener un ul. No es lo más común, pero es válido. Lo que no debes hacer es lo contrario sin sentido: meter tablas dentro de listas solo para acomodar el diseño.

¿Puedo poner una tabla dentro de un div? Claro que sí, y es lo más normal del mundo. El div agrupa la sección y la tabla vive adentro como parte del contenido. De hecho, es una buena práctica envolver tablas grandes en un div, porque luego con CSS puedes controlar cosas como el desplazamiento horizontal en pantallas chicas.

¿Cuántos niveles de listas anidadas puedo usar? Técnicamente no hay un límite en el estándar, pero en la práctica más de tres niveles se vuelve confuso para el lector. Si tu información necesita cuatro o cinco niveles de anidamiento, probablemente el problema es la organización de la información, no el HTML: considera dividirla en varias secciones.

¿El atributo type de ol sigue vigente o ya está obsoleto? Sigue vigente en HTML5. Hubo una época en que se consideró presentacional y se recomendaba hacerlo todo con CSS, pero el estándar actual lo mantiene como válido. Úsalo con confianza para casos simples; cuando necesites numeraciones muy personalizadas, CSS te dará más control.

¿Por qué mi tabla se ve sin líneas? Porque los navegadores no dibujan bordes en las tablas por defecto. No es un error: es el comportamiento estándar. Los bordes, los colores y el espaciado son responsabilidad de CSS. Resiste la tentación de usar el atributo border obsoleto; tu tabla está bien, solo está esperando su turno de maquillaje.

¿Puedo usar div dentro de un li? Sí, perfectamente. El li es un elemento que puede contener bloques, así que meter un div dentro de un li es válido. Es muy común en menús complejos o en listas de tarjetas, donde cada li contiene un div con la estructura de la tarjeta. Lo que no puedes hacer es meter un div dentro de un span o dentro de un a que esté dentro de un párrafo, porque ahí romperías la regla de que los elementos en línea no contienen bloques.

¿Span sirve para algo sin CSS? Casi nada visual, y está bien. El span es una herramienta de marcado: prepara el terreno para que CSS o JavaScript hagan su trabajo después. Un HTML

profesional está lleno de spans que "no se ven" pero que permiten que el diseño y la interactividad funcionen. No los veas como inútiles: son los ganchos invisibles de la página.

¿Cómo sé si algo debe ser lista, tabla o divs? Hazte tres preguntas. Primera: ¿son elementos hermanos del mismo grupo, como opciones o pasos? Entonces es lista (ul si el orden no importa, ol si importa). Segunda: ¿son datos con filas y columnas que se entienden mejor alineados, como precios u horarios? Entonces es tabla. Tercera: ¿es una zona o un componente de la página que agrupa cosas distintas, como una tarjeta o una sección? Entonces es div. Si ninguna de las tres te convence, probablemente necesitas una etiqueta semántica más específica (párrafo, encabezado, artículo), no un contenedor genérico.

Tu orden mental para construir páginas

Para cerrar la explicación, quiero dejarte un método de trabajo, un orden mental que puedes seguir cada vez que construyas una página. Los profesionales no escriben etiquetas al azar: primero piensan en la estructura y luego la traducen a HTML. Este es el orden que te recomiendo.

Paso uno: divide la página en grandes bloques. Mira tu diseño o tu idea y pregúntate cuántas zonas tiene: encabezado, contenido, barra lateral, pie. Cada zona será un div. No escribas nada más hasta tener claros esos bloques. Es como planear las habitaciones de una casa antes de comprar los muebles.

Paso dos: dentro de cada bloque, identifica los grupos. ¿Hay un menú? Es una lista no ordenada. ¿Hay pasos o un ranking? Es una lista ordenada. ¿Hay datos con filas y columnas? Es una tabla. ¿Hay un conjunto de tarjetas parecidas? Cada tarjeta es un div con su contenido adentro. En este paso traduces cada grupo de información a la herramienta correcta de la unidad.

Paso tres: marca los detalles en línea. Recorre el texto y detecta los fragmentos que necesitarán tratamiento especial: precios, fechas, nombres de marca, palabras a destacar, datos que JavaScript va a manipular. Cada uno de esos fragmentos lleva un span. No te preocupes todavía por cómo se van a ver: estás dejando los ganchos listos para CSS.

Paso cuatro: verifica la validez estructural. Repasa tu código con la lista de reglas de la unidad: cada li dentro de su ul u ol y bien cerrado, cada fila de tabla con el mismo número de celdas, caption en cada tabla, listas anidadas dentro del li, ningún bloque dentro de un elemento en línea, indentación consistente en todos los niveles. Esta revisión de dos minutos te ahorra horas de depuración después.

Paso cinco: pruébalo en el navegador y léelo como un usuario. ¿La información se entiende? ¿El menú parece menú? ¿La tabla se lee fácil? ¿Cada tarjeta se distingue como una unidad? Si algo se siente confuso, el problema casi siempre está en la agrupación: revisa si usaste la herramienta correcta para cada grupo.

Este método parece lento al principio, pero con la práctica se vuelve automático. En unas semanas vas a mirar cualquier diseño y ver de inmediato los divs, las listas y las tablas que lo componen, como un arquitecto que mira un edificio y ve la estructura detrás de la fachada. Esa visión estructural es lo que separa a quien "escribe etiquetas" de quien "construye páginas". Y se entrena exactamente así: bloque por bloque, grupo por grupo, detalle por detalle.

Ejemplos paso a paso

Ejemplo 1: menú de navegación con ul

Vamos a construir el menú de navegación de un sitio. Es el ejemplo más clásico y útil de lista no ordenada, y es exactamente lo que usan los sitios profesionales. Cada opción del menú es un li que contiene un enlace a.

```
<nav>
  <ul>
    <li><a href="index.html">Inicio</a></li>
    <li><a href="cursos.html">Cursos</a></li>
    <li><a href="tutoriales.html">Tutoriales</a></li>
    <li><a href="blog.html">Blog</a></li>
    <li><a href="contacto.html">Contacto</a></li>
  </ul>
</nav>
```

Analicemos línea por línea. La etiqueta nav es un contenedor semántico que indica "aquí hay navegación" (la verás a fondo en la unidad de HTML semántico, por ahora úsala como envoltura con significado). Dentro de nav va el ul: la lista de opciones. Cada li es una opción, y dentro de cada li hay un a con su href apuntando a la página correspondiente y su texto visible.

¿Por qué una lista y no cinco enlaces sueltos? Porque las opciones del menú forman un grupo: pertenecen a la misma navegación, están al mismo nivel y tienen el mismo propósito. La lista comunica esa relación. Además, cuando apliques CSS, vas a poder seleccionar todos los li de un jalón para quitarles las viñetas, ponerlos en horizontal y darles estilo de botones,

todo con unas pocas reglas. Si fueran enlaces sueltos sin lista, el CSS sería más complicado y menos ordenado.

Prueba este ejemplo en tu navegador: verás las cinco opciones en vertical con viñetas. Se ve simple, y está bien: la magia visual llega con CSS. Lo importante es que la estructura es la correcta y profesional. Como reto, agrega una sexta opción al menú y comprueba que no tienes que hacer nada más: la lista la acomoda sola.

Ejemplo 2: receta de cocina con ol y listas anidadas

Ahora un ejemplo que combina lista ordenada con listas no ordenadas anidadas: una receta de cocina. Los pasos van en ol porque el orden importa (no puedes hornear antes de mezclar), y dentro de algunos pasos anidamos un ul con detalles o ingredientes.

```
<h2>Receta: pan casero</h2>
<ol>
  <li>Mezcla los ingredientes secos
    <ul>
      <li>500 gramos de harina de trigo</li>
      <li>10 gramos de sal</li>
      <li>7 gramos de levadura seca</li>
    </ul>
  </li>
  <li>Agrega 300 mililitros de agua tibia y amasa 10 minutos</li>
  <li>Deja reposar la masa 1 hora en un lugar tibio
    <ul>
      <li>La masa debe duplicar su tamaño</li>
      <li>Tápala con un trapo limpio y húmedo</li>
    </ul>
  </li>
  <li>Hornea a 200 grados durante 30 minutos</li>
  <li>Deja enfriar antes de cortar</li>
</ol>
```

Observa con lupa la estructura de anidamiento. El paso 1 es un li que contiene texto ("Mezcla los ingredientes secos") y, dentro del mismo li, un ul con los tres ingredientes. El ul anidado se abre después del texto del li y se cierra antes de que se cierre el li. Lo mismo ocurre en el paso 3 con las notas del reposado. Los pasos 2, 4 y 5 son li simples con solo texto.

El navegador muestra los pasos numerados del 1 al 5, y los detalles anidados con viñetas y sangría, dejando clarísimo qué detalle pertenece a qué paso. Esta es la forma profesional de

escribir procedimientos con subdetalles: tutoriales, manuales de armado, instrucciones de instalación de software. Fíjate en que mezclamos ol (pasos) con ul (detalles): cada tipo de lista hace el trabajo para el que fue diseñada.

Como ejercicio de modificación, cambia el ol a type="A" y observa cómo los pasos se identifican con letras. Luego prueba start="3" para ver cómo continúa una numeración. Estos experimentos te enseñan más que leer diez páginas de teoría.

Ejemplo 3: tabla de datos con caption

Construyamos una tabla completa y bien formada: el registro de calificaciones de un grupo. Incluye caption, fila de encabezados con th y filas de datos con td.

```
<table>
  <caption>Calificaciones del grupo A, parcial 1</caption>
  <tr>
    <th>Estudiante</th>
    <th>HTML</th>
    <th>CSS</th>
    <th>Promedio</th>
  </tr>
  <tr>
    <td>Ana</td>
    <td>95</td>
    <td>88</td>
    <td>91.5</td>
  </tr>
  <tr>
    <td>Luis</td>
    <td>80</td>
    <td>92</td>
    <td>86</td>
  </tr>
  <tr>
    <td>Marta</td>
    <td>90</td>
    <td>90</td>
    <td>90</td>
  </tr>
</table>
```

Vamos por partes. El caption va primero, como primer hijo de table: "Calificaciones del grupo A, parcial 1". Le dice a cualquiera que vea la tabla de qué trata, sin tener que adivinar. Luego la fila de encabezados: cuatro th que describen las cuatro columnas (Estudiante, HTML, CSS, Promedio). El navegador los muestra en negritas y centrados, y los lectores de pantalla los usan para anunciar cada dato con su columna.

Después vienen las tres filas de datos, cada una con cuatro td: el nombre del estudiante y sus tres calificaciones. Cuenta las celdas de cada fila: todas tienen exactamente cuatro. Esa regularidad es lo que mantiene la cuadrícula perfecta. Si por accidente una fila tuviera tres celdas, la tabla se descuadraría y el descuadre sería visible de inmediato.

Prueba a agregar una cuarta estudiante copiando una fila completa de tr y cambiando los datos. Luego prueba a agregar una columna nueva, por ejemplo "JavaScript": tendrás que agregar un th en la fila de encabezados y un td en cada fila de datos. Este ejercicio te enseña la disciplina de las tablas: los cambios se hacen fila por fila, celda por celda, manteniendo siempre la regularidad.

Nota que la tabla se ve sin bordes por defecto. Es normal: los bordes son trabajo de CSS. No caigas en la tentación de usar el atributo border obsoleto; cuando llegues a la unidad de CSS vas a aprender a ponerle a esta misma tabla bordes elegantes, colores alternados por fila y encabezados con fondo de color, todo sin tocar el HTML.

Ejemplo 4: tarjeta de perfil con div y span

Ahora combinemos div y span en algo muy real: una tarjeta de perfil como las que ves en redes sociales o en páginas de equipo. El div agrupa toda la tarjeta y los spans destacan fragmentos dentro del texto.

```
<div>
  <h3>Esteban</h3>
  <p>Desarrollador web y fundador de <span>Lucas EGA Academy</span>.</p>
  <p>Especialidades:</p>
  <ul>
    <li>HTML5 y CSS</li>
    <li>JavaScript</li>
    <li>Python</li>
  </ul>
  <p>Contacto: <span>hola@lucasegaacademy.com</span></p>
</div>
```

Analicemos la estructura. El div exterior es la caja de mudanza: todo lo de la tarjeta vive adentro, formando una unidad lógica. Dentro hay un h3 con el nombre, un párrafo de presentación donde un span marca "Lucas EGA Academy" (ese fragmento podrá llevar un estilo distinto, como un color de marca), otro párrafo que introduce la lista, un ul con las tres especialidades, y un párrafo final de contacto donde otro span marca el correo electrónico.

Fíjate en cómo conviven todos los elementos de la unidad en un solo ejemplo: el div agrupa, el span destaca fragmentos en línea, y el ul organiza las especialidades. Así se construyen las páginas reales: combinando las herramientas. Esta tarjeta, con unas reglas de CSS (borde redondeado, sombra, fondo claro, el span de la academia en color azul y el correo en negritas), se vería como una tarjeta profesional de cualquier sitio moderno. Y todo empieza con este HTML bien estructurado.

Como reto, crea dos tarjetas más copiando el div completo y cambiando los datos: inventa dos compañeros de equipo con sus especialidades. Verás lo fácil que es replicar componentes cuando están bien agrupados en divs: copias la caja completa y solo cambias el contenido.

Práctica

Ejercicio 1: lista de compras del súper

Enunciado: crea una página HTML completa (con doctype, html, head, body y un h1 con el título) que contenga una lista no ordenada con al menos 8 productos del súper, organizados en 3 categorías usando listas anidadas: Frutas, Lácteos y Panadería, cada una con sus productos.

Solución explicada paso a paso:

Paso 1: escribe la estructura base del documento. Todo ejemplo de esta unidad parte de un HTML válido:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mi lista del súper</title>
</head>
<body>
  <h1>Lista del súper</h1>
</body>
</html>
```

Paso 2: dentro del body, después del h1, abre el ul principal que contendrá las 3 categorías. Cada categoría será un li.

Paso 3: dentro del li de cada categoría, escribe el nombre de la categoría y abre un ul anidado con sus productos. Recuerda la regla de oro: el ul anidado va dentro del li, antes de su cierre.

Paso 4: verifica que cada li esté cerrado y que cada nivel tenga su indentación. El resultado final:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mi lista del súper</title>
</head>
<body>
  <h1>Lista del súper</h1>
  <ul>
    <li>Frutas
      <ul>
        <li>Manzanas</li>
        <li>Plátanos</li>
        <li>Naranjas</li>
      </ul>
    </li>
    <li>Lácteos
      <ul>
        <li>Leche</li>
        <li>Queso</li>
        <li>Yogur</li>
      </ul>
    </li>
    <li>Panadería
      <ul>
        <li>Bolillos</li>
        <li>Tortillas de harina</li>
      </ul>
    </li>
  </ul>
</body>
</html>
```

Qué aprendiste: a construir listas anidadas con la estructura correcta (la lista hija dentro del li), a organizar información por categorías y a mantener la indentación en varios niveles. Abre el archivo en tu navegador y comprueba que las categorías se ven con una viñeta y los productos con otra, con sangría.

Ejercicio 2: instructivo numerado con letras

Enunciado: crea una lista ordenada con los 5 pasos para crear una cuenta de correo, pero usando letras mayúsculas (type="A") en lugar de números, y haz que la lista empiece en la letra C (start="3"). Además, en el paso 2 anida una lista no ordenada con 3 consejos de seguridad para la contraseña.

Solución explicada paso a paso:

Paso 1: abre el ol con los dos atributos combinados: type="A" para letras mayúsculas y start="3" para empezar en la tercera letra, que es la C.

Paso 2: escribe los 5 li con los pasos: elegir proveedor, crear contraseña, verificar identidad, configurar recuperación y listo.

Paso 3: dentro del li del paso 2, antes de cerrarlo, anida un ul con los 3 consejos: usar más de 12 caracteres, mezclar mayúsculas y números, no reutilizar contraseñas.

Paso 4: cierra todo correctamente. El código final:

```
<ol type="A" start="3">
  <li>Elige un proveedor de correo (Gmail, Outlook, etc.)</li>
  <li>Crea una contraseña segura
    <ul>
      <li>Usa más de 12 caracteres</li>
      <li>Mezcla mayúsculas, minúsculas y números</li>
      <li>No reutilices contraseñas de otros sitios</li>
    </ul>
  </li>
  <li>Verifica tu identidad con tu teléfono</li>
  <li>Configura un correo de recuperación</li>
  <li>Listo: ya puedes enviar y recibir mensajes</li>
</ol>
```

Qué aprendiste: a combinar los atributos type y start (la lista muestra C, D, E, F, G), a mezclar ol con ul anidado, y a ver cómo el navegador recalcula la identificación de cada ítem automáticamente. Prueba cambiar start a 1 y type a "i" para ver los números romanos en minúsculas.

Ejercicio 3: tabla de precios de una cafetería

Enunciado: crea una tabla con el menú de precios de una cafetería: columnas Producto, Tamaño Chico, Tamaño Mediano y Tamaño Grande; 4 productos (café americano, capuchino, chocolate caliente y té chai) con sus precios inventados. Incluye caption y usa th para los encabezados.

Solución explicada paso a paso:

Paso 1: abre table y escribe el caption como primer hijo: "Precios de cafetería El Grano, 2026".

Paso 2: escribe la fila de encabezados con 4 th: Producto, Chico, Mediano, Grande.

Paso 3: escribe las 4 filas de datos, cada una con 4 td: el nombre del producto y sus 3 precios. Cuenta las celdas de cada fila mientras escribes para mantener la regularidad.

Paso 4: revisa que todas las etiquetas estén cerradas. El código final:

```

<table>
  <caption>Precios de cafetería El Grano, 2026</caption>
  <tr>
    <th>Producto</th>
    <th>Chico</th>
    <th>Mediano</th>
    <th>Grande</th>
  </tr>
  <tr>
    <td>Café americano</td>
    <td>$25</td>
    <td>$30</td>
    <td>$35</td>
  </tr>
  <tr>
    <td>Capuchino</td>
    <td>$35</td>
    <td>$42</td>
    <td>$48</td>
  </tr>
  <tr>
    <td>Chocolate caliente</td>
    <td>$30</td>
    <td>$38</td>
    <td>$45</td>
  </tr>
  <tr>
    <td>Té chai</td>
    <td>$32</td>
    <td>$40</td>
    <td>$46</td>
  </tr>
</table>

```

Qué aprendiste: a construir una tabla realista de 5 filas por 4 columnas, a usar caption y th correctamente, y la disciplina de mantener el mismo número de celdas en cada fila. Como reto extra, agrega una quinta columna "Tamaño Extragrande": recuerda que debes agregar un th y un td en cada fila.

Ejercicio 4: tarjeta de curso con div y span

Enunciado: crea una tarjeta de curso usando un div como contenedor. Dentro del div incluye: un h2 con el nombre del curso, un párrafo con la descripción donde uses un span para destacar la palabra "gratis", una lista no ordenada con 3 temas que se ven en el curso, y un párrafo final con un span que marque el precio "\$0". Todo debe estar dentro del mismo div.

Solución explicada paso a paso:

Paso 1: abre el div contenedor. Todo lo demás vivirá adentro.

Paso 2: escribe el h2 con el nombre del curso, por ejemplo "HTML5 desde cero".

Paso 3: escribe el párrafo de descripción e incluye el span alrededor de la palabra "gratis": "Aprende a crear páginas web gratis y a tu ritmo."

Paso 4: escribe el ul con los 3 temas: estructura de un documento, listas y tablas, formularios básicos.

Paso 5: escribe el párrafo del precio con el span: "Precio: \$0 para siempre."

Paso 6: cierra el div. El código final:

```
<div>
  <h2>HTML5 desde cero</h2>
  <p>Aprende a crear páginas web <span>gratis</span> y a tu ritmo, sin
conocimientos previos.</p>
  <ul>
    <li>Estructura de un documento HTML</li>
    <li>Listas, tablas y contenedores</li>
    <li>Formularios básicos</li>
  </ul>
  <p>Precio: <span>$0</span> para siempre.</p>
</div>
```

Qué aprendiste: a usar div como caja contenedora de un componente completo, a marcar fragmentos en línea con span sin romper el flujo del texto, y a combinar en un solo bloque las tres herramientas de la unidad (div, span y ul). Imagina esta tarjeta con CSS: el div con borde y sombra, los spans en color verde destacando "gratis" y "\$0". El HTML ya está listo para eso.

Ejercicio 5: página completa que combine todo

Enunciado: crea una página HTML completa para "Mi primera chamba como desarrollador" que combine todo lo de la unidad: un div contenedor principal; dentro, un h1 con el título; un párrafo de presentación con un span destacando tu nombre; una lista no ordenada con 4 habilidades; una lista ordenada con los 3 pasos de tu plan (aprender HTML, practicar todos los días, crear portafolio); y una tabla con caption de 2 columnas (Habilidad, Nivel) y 3 filas de datos.

Solución explicada paso a paso:

Paso 1: escribe la estructura base del documento con doctype, html, head (con title) y body.

Paso 2: dentro del body abre el div contenedor principal que agrupará toda la página.

Paso 3: dentro del div, escribe el h1 y el párrafo de presentación con el span para tu nombre.

Paso 4: agrega el ul con las 4 habilidades (HTML5, CSS, JavaScript, Git).

Paso 5: agrega el ol con los 3 pasos del plan.

Paso 6: agrega la tabla: caption "Mi nivel actual", fila de encabezados con 2 th, y 3 filas de datos con 2 td cada una.

Paso 7: cierra el div, el body y el html. Revisa la indentación de todo el documento. El código final:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mi primera chamba</title>
</head>
<body>
  <div>
    <h1>Mi camino a mi primera chamba</h1>
    <p>Hola, soy <span>Esteban</span> y estoy aprendiendo desarrollo web para conseguir mi primer trabajo en tecnología.</p>
    <h2>Mis habilidades</h2>
    <ul>
      <li>HTML5</li>
      <li>CSS</li>
      <li>JavaScript</li>
      <li>Git</li>
    </ul>
    <h2>Mi plan</h2>
```

```

<ol>
  <li>Aprender HTML a fondo</li>
  <li>Practicar todos los días</li>
  <li>Crear mi portafolio</li>
</ol>
<h2>Mi nivel actual</h2>
<table>
  <caption>Autoevaluación de habilidades, septiembre 2026</caption>
  <tr>
    <th>Habilidad</th>
    <th>Nivel</th>
  </tr>
  <tr>
    <td>HTML5</td>
    <td>Intermedio</td>
  </tr>
  <tr>
    <td>CSS</td>
    <td>Básico</td>
  </tr>
  <tr>
    <td>JavaScript</td>
    <td>Básico</td>
  </tr>
</table>
</div>
</body>
</html>

```

Qué aprendiste: a orquestar todos los elementos de la unidad en una página real y coherente. El div agrupa todo, los spans destacan fragmentos, las listas organizan habilidades y pasos, y la tabla presenta datos tabulares con caption y encabezados. Esta es la estructura mental que vas a usar en cada página que construyas de aquí en adelante: primero agrupas con divs, luego organizas con listas y tablas, y finalmente marcas detalles con spans. Si lograste este ejercicio sin copiar, ya dominas la unidad.

Resumen de la unidad

- Las listas agrupan elementos relacionados: ul es la lista no ordenada con viñetas (el orden no importa) y ol es la lista ordenada con números (el orden sí importa).

- Cada elemento de una lista se marca con li, siempre dentro de un ul o un ol, y siempre con su etiqueta de cierre.
- La etiqueta ol acepta los atributos type (1, a, A, i, I para cambiar el estilo de numeración), start (número o letra inicial) y reversed (cuenta hacia atrás).
- Las listas anidadas representan jerarquías: la lista hija siempre va dentro del li del padre, antes de que se cierre, nunca como hermana.
- Los menús de navegación, índices, pasos de tutoriales e ingredientes de recetas son listas en el HTML profesional; usar listas mejora la accesibilidad porque los lectores de pantalla anuncian la relación entre los elementos.
- Las tablas presentan datos tabulares en filas y columnas: table es el contenedor, tr es cada fila, th es la celda de encabezado y td es la celda de dato.
- Cada fila de una tabla debe tener el mismo número de celdas para que la cuadrícula quede pareja; los th se muestran en negritas y centrados y ayudan a los lectores de pantalla a relacionar cada dato con su encabezado.
- La etiqueta caption va como primer hijo de table y le da un título descriptivo a la tabla: mejora la claridad y la accesibilidad, úsala siempre.
- Las tablas son solo para datos tabulares, nunca para maquetar el diseño de la página; el atributo border está obsoleto en HTML5 y los bordes se hacen con CSS.
- El div es un contenedor de bloque: ocupa todo el ancho, empieza en línea nueva y agrupa secciones, tarjetas y áreas de la página como una caja de mudanza.
- El span es un contenedor en línea: ocupa solo el espacio de su contenido, no rompe la línea y marca fragmentos de texto como un resaltador.
- Los elementos de bloque pueden contener bloques y elementos en línea, pero los elementos en línea no deben contener bloques.
- Usa div para agrupar bloques con propósito y span para marcar fragmentos con propósito; evita la divitis (usar divs para todo) y prefiere las etiquetas semánticas cuando exista una más específica.
- La indentación consistente es obligatoria con listas anidadas, tablas y divs anidados: es la única forma de no perderte y de escribir HTML que otros puedan leer.
- El poder real está en combinar: tarjetas (div) con listas de características (ul), secciones (div) con tablas de datos (table), menús (ul) dentro de encabezados (div).

UNIDAD 4: Atributos y formularios

Bienvenido a la cuarta unidad del Curso Completo de HTML5 de Lucas EGA Academy. Hasta ahora has caminado un camino bien sólido: en la Unidad 1 aprendiste qué es HTML y cómo se escribe una etiqueta; en la Unidad 2 viste la estructura completa de una página web y la importancia del encabezado; en la Unidad 3 conociste las etiquetas que dan vida al contenido, como los párrafos, los encabezados, las listas, los enlaces y las imágenes. Todo eso te convirtió en alguien capaz de publicar una página que se ve y se lee bien.

Pero aquí viene la verdad importante: una página sin atributos y sin formularios es como una casa sin puertas ni ventanas. Puedes verla desde afuera, puedes leer lo que dice el letrero, pero no puedes entrar, ni tocar, ni platicar con quien vive adentro. Los atributos son los que le dan personalidad, identidad y comportamiento a cada etiqueta. Los formularios son los que abren la puerta para que el visitante de tu página deje de ser un simple espectador y se convierta en participante: puede escribirte, registrarse, responder una encuesta, pedir un producto o dejar un comentario.

En esta unidad vas a aprender dos temas que trabajan juntos todo el tiempo: los atributos y los formularios. Primero vas a entender qué son los atributos, esos pedacitos de información extra que se escriben dentro de la etiqueta de apertura, y cómo cada uno tiene una misión distinta. Vas a conocer los atributos globales, que son los que puedes usar en casi cualquier etiqueta: `id`, `class`, `title`, `style`, `lang`, `data-*`, `hidden` y `tabindex`. Vas a aprender la diferencia entre `id` y `class`, que es una de las confusiones más comunes entre quienes empiezan, y te vas a quedar con una regla de oro que no vas a olvidar nunca.

Después vas a entrar de lleno al mundo de los formularios. Vas a ver cómo funciona la etiqueta `form`, qué son los atributos `action` y `method`, y por qué la diferencia entre `get` y `post` es algo que todo desarrollador web debe entender desde el primer día. Vas a conocer la familia completa de los campos: `input` en todas sus versiones, `textarea` para textos largos, `select` y `option` para listas desplegables, `button` para los botones de acción y `label` para que tus formularios sean accesibles y fáciles de usar. También vas a descubrir los `inputs` inteligentes que trajo HTML5: `email` que valida correos solo, `password` que oculta lo que escribes, `date` que abre un calendario, `color` que abre un selector de color, y muchos más.

¿Por qué importa tanto esta unidad? Porque aquí es donde tu página deja de ser un folleto y se convierte en una herramienta. Piensa en las páginas que usas todos los días: el buscador de Google es un formulario; el inicio de sesión de tu correo es un formulario; la caja de comentarios de un video es un formulario; el botón de comprar de una tienda es un formulario.

Todo lo interactivo en la web empieza con lo que vas a aprender aquí. Sin formularios no hay registros, no hay contactos, no hay ventas, no hay comunidad. Dominar este tema te pone un paso adelante de muchísima gente que solo sabe poner texto e imágenes.

¿Cómo encaja esta unidad en el curso? Es el puente entre el HTML estático que ya dominas y todo lo que viene después. En las próximas unidades vas a darle diseño a tus páginas con CSS, y adivina qué: los atributos id y class son exactamente los ganchos que CSS usa para pintar y acomodar cada cosa. Sin ellos, el diseño sería imposible. Más adelante, cuando llegues a JavaScript, los formularios serán tu campo de juego para validar datos, mostrar mensajes y hacer páginas que reaccionan. Todo lo que aprendas aquí lo vas a usar una y otra vez, en cada proyecto, en cada curso, en cada página que construyas de aquí en adelante. Así que tómallo con calma, practica cada ejemplo y no te saltes los ejercicios: esta unidad es de las que más retorno te dan por cada hora que le inviertes.

Explicación detallada

Qué son los atributos

Imagina que las etiquetas HTML son personas en una fiesta. El nombre de la etiqueta te dice quién es cada quien: 'esta es un párrafo', 'esta es una imagen', 'este es un botón'. Pero solo con el nombre no sabes mucho más: ¿cómo se llama?, ¿de qué color va vestido?, ¿a qué grupo pertenece?, ¿qué hace si le hablas? Los atributos son como las etiquetas de equipaje que cada persona trae colgadas: pedacitos de información extra que describen, identifican y modifican a cada etiqueta.

Un atributo siempre vive dentro de la etiqueta de apertura, y siempre tiene la misma forma: un nombre, un signo de igual y un valor entre comillas. Mira este ejemplo:

```
<p title='Este es un párrafo especial'>Hola, mundo</p>
```

Aquí la etiqueta es p, el atributo es title y su valor es 'Este es un párrafo especial'. Cuando el visitante pase el mouse sobre ese párrafo, el navegador le va a mostrar ese texto como una ayudita emergente. El contenido visible sigue siendo 'Hola, mundo', pero la etiqueta ahora trae información extra que el navegador sabe interpretar.

Los atributos nunca se escriben en la etiqueta de cierre. Esto es un error clásico de principiante:

```
<!-- MAL: el atributo va en la apertura, no en el cierre -->
<p>Hola</p title='ayuda'>
<!-- BIEN -->
<p title='ayuda'>Hola</p>
```

Otra regla importante: los valores de los atributos van entre comillas. En este libro usamos comillas simples, pero también puedes usar comillas dobles. Lo importante es que abras y cierres con el mismo tipo. Si mezclas, el navegador se confunde y el atributo no funciona.

Los atributos son opcionales: una etiqueta funciona sin ellos. Pero sin atributos tus etiquetas son genéricas, todas iguales, sin identidad. Con atributos les das superpoderes: puedes identificarlas una por una, agruparlas, darles estilo, esconderlas, hacerlas accesibles y conectarlas con otras tecnologías. En esta unidad vas a conocer primero los atributos globales, que son los comodines que funcionan en casi todas las etiquetas, y después los atributos propios de los formularios.

Atributos globales: los comodines de HTML

Se les llama globales porque no pertenecen a una etiqueta en especial: los puedes usar en un párrafo, en una imagen, en un div, en un botón, en casi todo. Son como la navaja suiza de HTML: unas cuantas herramientas que resuelven muchísimos problemas. Vamos a verlos uno por uno, con ejemplos que puedes probar en tu navegador.

El atributo id: tu credencial única

El atributo id le pone un nombre único a un elemento. Único significa único de verdad: en toda la página solo puede existir un elemento con ese id. Es como tu CURP o tu número de identificación oficial: nadie más en el país tiene el tuyo, y sirve para señalarte a ti sin confusión.

```
<h1 id='titulo-principal'>Mi tienda en línea</h1>
<p id='descripcion-tienda'>Vendemos playeras desde 2020.</p>
```

¿Para qué sirve el id? Para tres cosas muy importantes. Primera: es el gancho que usa CSS para darle estilo a un elemento específico, sin afectar a los demás. Segunda: es la forma en que JavaScript encuentra un elemento para interactuar con él, como cuando quieres que al presionar un botón pase algo. Tercera: te permite crear enlaces que saltan directamente a una sección de la página, usando el símbolo de gato. Por ejemplo, si tienes `<h2 id='contacto'>Contacto</h2>`, un enlace como `<a href='#contacto'>Ir a contacto</a>` lleva al visitante directo a esa sección.

Reglas del id que debes respetar siempre:

- Solo un elemento por página puede tener un id determinado. Si repites el mismo id en dos elementos, el navegador no te va a marcar error, pero CSS y JavaScript se van a comportar de forma rara e impredecible. Es uno de los errores más comunes y más difíciles de detectar.
- El id no puede empezar con un número y no debe llevar espacios. Usa guiones o guiones bajos para separar palabras: 'título-principal' o 'título_principal'.
- El id distingue entre mayúsculas y minúsculas: 'Menú' y 'menú' son dos ids distintos. Para evitar dolores de cabeza, usa siempre minúsculas.

El atributo class: el uniforme del equipo

Si el id es tu credencial única, el atributo class es el uniforme de tu equipo: sirve para marcar a varios elementos como parte del mismo grupo. Muchos elementos pueden compartir la misma clase, y un solo elemento puede tener varias clases separadas por espacios.

```
<p class='aviso'>Oferta válida solo hoy.</p>
<p class='aviso'>Envío gratis en compras mayores a 500 pesos.</p>
<p class='aviso importante'>Últimas piezas disponibles.</p>
```

Aquí los tres párrafos pertenecen al grupo 'aviso', y el tercero además pertenece al grupo 'importante'. Con CSS después podrás decir 'todos los elementos de la clase aviso se ven así', y de un solo golpe les das estilo a los tres. Esa es la magia de class: te ahorra repetir trabajo.

La diferencia entre id y class, de una vez por todas

Esta es la confusión número uno de los principiantes, así que vamos a dejarla clarísima con una analogía. Piensa en una escuela. Cada alumno tiene un número de matrícula único: ese es el id. Nadie más tiene ese número, y sirve para encontrar su expediente exacto. Pero los alumnos también pertenecen a grupos: el grupo 3A, el equipo de fútbol, el club de lectura. Esos grupos son las clases: muchos alumnos comparten el mismo grupo, y un alumno puede estar en varios grupos a la vez.

Regla de oro para decidir cuál usar:

- ¿Necesitas señalar a UN elemento específico, uno solo, para darle un estilo único o para que JavaScript lo encuentre? Usa id.

- ¿Necesitas aplicar lo mismo a VARIOS elementos, o crees que en el futuro habrá más elementos iguales? Usa class.

En la práctica vas a usar class muchísimo más que id. El id es para casos especiales: el encabezado principal, el menú de navegación, el pie de página, el formulario de contacto. La clase es para lo repetitivo: tarjetas de productos, avisos, botones, párrafos destacados. Cuando dudes, pregunta: '¿esto se va a repetir?' Si la respuesta es sí, es class.

Mira este ejemplo completo donde conviven los dos:

```
<header id='encabezado-sitio'>
  <h1 class='titulo-grande'>Lucas EGA Academy</h1>
  <p class='subtitulo'>Aprende tecnología en español</p>
</header>
<section class='tarjeta-curso'>
  <h2 class='titulo-curso'>HTML5 desde cero</h2>
  <p class='descripcion-curso'>Crea tus primeras páginas web.</p>
</section>
<section class='tarjeta-curso'>
  <h2 class='titulo-curso'>CSS para principiantes</h2>
  <p class='descripcion-curso'>Dale diseño a tus páginas.</p>
</section>
```

El header tiene un id porque solo hay un encabezado del sitio. Las secciones comparten la clase 'tarjeta-curso' porque son varias tarjetas iguales. Más adelante, con una sola regla de CSS podrás darles el mismo diseño a todas las tarjetas.

El atributo title: la ayuda emergente

El atributo title muestra un mensajito cuando el visitante deja el mouse quieto sobre el elemento. Es como esos letreros de ayuda que aparecen en los programas de computadora.

```
<p title='Pasa el mouse para ver este mensaje'>Párrafo con ayuda</p>
<img src='foto.jpg' alt='Foto' title='Foto tomada en 2024'>
<abbr title='Lenguaje de Marcado de Hipertexto'>HTML</abbr>
```

Un uso muy elegante es con la etiqueta abbr para explicar siglas: el visitante ve 'HTML' y al pasar el mouse descubre qué significa. También sirve en enlaces para dar una pista de a dónde llevan, y en botones para explicar qué hacen. No abuses de él: si la información es importante, mejor escríbela visible en la página, porque en celulares no hay mouse y el title no se puede ver.

El atributo style: estilos en línea

El atributo style te permite escribir estilos CSS directamente en la etiqueta, sin necesidad de una hoja de estilos separada. Es rápido y directo, perfecto para pruebas y para cambios pequeños.

```
<p style='color: red;'>Este texto sale en rojo.</p>
<p style='color: blue; font-size: 20px;'>Texto azul y más grande.</p>
```

Cada estilo se escribe como propiedad, dos puntos, valor y punto y coma. Puedes poner varios estilos separados por punto y coma. Pero ojo con una advertencia importante: el atributo style solo debe usarse para pruebas rápidas o casos muy puntuales. Si empiezas a poner style en todas tus etiquetas, tu código se vuelve un caos difícil de mantener. La buena práctica es usar class y escribir los estilos en un archivo CSS aparte, tema que verás a fondo en el curso de CSS de Lucas EGA Academy. Piensa en style como la cinta adhesiva: sirve para una reparación rápida, pero no para construir la casa.

El atributo lang: el idioma del contenido

El atributo lang le dice al navegador y a los lectores de pantalla en qué idioma está escrito el contenido. Lo más común es verlo en la etiqueta html al inicio de cada página:

```
<html lang='es'>
```

Eso significa 'toda esta página está en español'. Pero también puedes usarlo en partes pequeñas cuando cambias de idioma dentro de la página:

```
<p>La palabra <span lang='en'>computer</span> significa computadora.</p>
```

¿Por qué importa? Porque los lectores de pantalla que usan las personas con discapacidad visual necesitan saber el idioma para pronunciar bien las palabras. También ayuda a los traductores automáticos y a los buscadores a entender tu contenido. Es un detalle pequeño que hace tu página más profesional y más accesible.

El atributo data-*: tu bodega privada de datos

Los atributos data-* te permiten guardar información extra en cualquier elemento, con el nombre que tú quieras, siempre que empiece con 'data-'. El navegador los ignora por completo, pero JavaScript puede leerlos después. Es como pegarle una notita adhesiva a una caja: tú sabes qué dice y para qué sirve.

```
<button data-producto-id='42' data-precio='299'>Agregar al carrito</button>  
<div data-usuario='ana123' data-nivel='avanzado'>Perfil de Ana</div>
```

En el primer ejemplo, el botón guarda el id del producto y su precio. Cuando el visitante presione el botón, JavaScript puede leer esos datos y saber exactamente qué producto agregar al carrito, sin tener que buscar la información en otro lado. Tú inventas los nombres después del guion: data-precio, data-color, data-talla, lo que necesites. Es una de las herramientas favoritas de los desarrolladores porque mantiene los datos pegados al elemento al que pertenecen.

El atributo hidden: el truco de magia

El atributo hidden esconde un elemento: no se muestra en la página, como si no existiera. No necesita valor, con solo escribirlo basta.

```
<p hidden>Este párrafo está escondido.</p>  
<p>Este párrafo sí se ve.</p>
```

¿Para qué querrías esconder algo? Mucho más de lo que imaginas. Puedes tener mensajes de error escondidos que solo aparecen cuando el usuario hace algo mal, secciones que se muestran al presionar un botón, o contenido que JavaScript va a revelar en el momento adecuado. Es la base de muchísimas interacciones: todo lo que aparece y desaparece en una página moderna empezó con algo escondido.

La diferencia con borrar el elemento es que el elemento sigue existiendo en el código, listo para mostrarse cuando se necesite. JavaScript puede quitar el atributo hidden en cualquier momento y el contenido aparece como por arte de magia.

El atributo tabindex: el orden del teclado

Muchas personas navegan la web usando solo el teclado, presionando la tecla Tab para saltar de un elemento a otro. El atributo tabindex controla ese orden. Con tabindex='0' incluyes un elemento en el orden natural de navegación; con un número positivo lo pones antes que otros.

```
<button tabindex='1'>Primero</button>
<button tabindex='2'>Segundo</button>
<div tabindex='0'>Este div también se puede seleccionar con Tab</div>
```

La recomendación de los expertos es sencilla: usa `tabindex='0'` cuando quieras que algo sea alcanzable con el teclado, y evita los números positivos porque rompen el orden lógico de la página y confunden a quienes navegan con teclado. Es un atributo que casi no vas a escribir a mano, pero debes saber que existe porque la accesibilidad con teclado es parte de hacer páginas para todas las personas.

Los formularios: la puerta de entrada de tu página

Si los atributos son las etiquetas de equipaje, un formulario es la hoja de registro que te dan al entrar a un edificio: tiene espacios en blanco donde escribes tu nombre, tu correo, el motivo de tu visita, y al final la entregas en recepción. En la web funciona igual: el formulario es un conjunto de campos donde el visitante escribe información, y un botón que envía esos datos a algún destino.

Todo formulario empieza con la etiqueta `form`, que actúa como la carpeta que guarda todas las hojas juntas. Dentro de `form` van los campos y los botones. Mira el formulario más sencillo posible:

```
<form>
  <input type='text' placeholder='Escribe tu nombre'>
  <button>Enviar</button>
</form>
```

La etiqueta `form` es el contenedor. La etiqueta `input` crea un campo donde el usuario escribe: `type='text'` significa que es un campo de texto normal. El atributo `placeholder` pone un texto de ejemplo dentro del campo, que desaparece cuando el usuario empieza a escribir. Y `button` crea el botón para enviar. Con solo estas tres etiquetas ya tienes un formulario funcional en apariencia. Pero para que realmente haga algo al enviarse, `form` necesita dos atributos fundamentales: `action` y `method`.

action y method: a dónde van los datos y cómo viajan

El atributo `action` dice a dónde se envían los datos del formulario. Su valor normalmente es la dirección de un programa en el servidor que va a recibir y procesar la información, por ejemplo guardar un mensaje de contacto o registrar a un usuario nuevo.

```
<form action='/guardar-mensaje'>
```

Si no pones action, el formulario se envía a la misma página donde está, lo cual casi nunca es lo que quieres.

El atributo method dice cómo viajan los datos, y tiene dos valores posibles: get y post. Esta diferencia es importantísima y vale la pena entenderla bien desde ahora.

Con method='get', los datos viajan pegados en la dirección web, a la vista de todos. Si un formulario pide tu nombre y escribes 'Ana', al enviarse la dirección se vería algo así: /buscar?nombre=Ana. Todo el mundo puede ver esos datos: quedan en el historial del navegador, se pueden copiar y pegar, y aparecen en los registros. Por eso get se usa solo para cosas sin importancia secreta, como un buscador: cuando buscas algo en Google, la búsqueda viaja en la dirección y eso está bien, hasta es útil porque puedes compartir el enlace con los resultados.

Con method='post', los datos viajan escondidos dentro del mensaje que el navegador envía al servidor, sin aparecer en la dirección web. Es como mandar una carta en un sobre cerrado en vez de escribir el mensaje en una postal que todos pueden leer. Por eso post se usa para todo lo delicado o importante: contraseñas, datos personales, mensajes de contacto, registros, compras. Regla sencilla: si la información es privada o si el envío crea o cambia algo (registrar un usuario, publicar un comentario, hacer una compra), usa post. Si solo estás pidiendo información sin cambiar nada (una búsqueda, un filtro), get está bien.

```
<!-- Formulario de búsqueda: get está bien -->
<form action='/buscar' method='get'>
  <input type='search' name='q' placeholder='Buscar en el sitio'>
  <button>Buscar</button>
</form>

<!-- Formulario de registro: post es lo correcto -->
<form action='/registrar' method='post'>
  <input type='text' name='nombre' placeholder='Tu nombre'>
  <input type='email' name='correo' placeholder='Tu correo'>
  <button>Registrarme</button>
</form>
```

Fíjate que en estos ejemplos apareció un atributo nuevo: name. El atributo name es el nombre con el que viaja cada dato. Sin name, el campo es como una hoja de registro sin título: el servidor recibe el dato pero no sabe qué significa. Cuando el formulario se envía, los datos

viajan en pares de nombre y valor, por ejemplo nombre=Ana y correo=ana@ejemplo.com. Si olvidas poner name en un input, ese campo simplemente no se envía: es como si no existiera. Este es uno de los errores más comunes y frustrantes: el formulario se ve perfecto, el botón funciona, pero al servidor no le llega nada porque faltó el name.

input: el campo multiusos

La etiqueta input es la más versátil de los formularios: con solo cambiar su atributo type se transforma en cosas muy distintas. No tiene etiqueta de cierre porque no lleva contenido adentro: es un elemento vacío, como img.

Ya viste type='text', el campo de texto normal. Pero input tiene una familia enorme de tipos, y HTML5 agregó varios inteligentes que hacen trabajo por ti. Los veremos todos en la siguiente subsección. Por ahora, conoce los atributos que casi todos los inputs comparten:

- name: el nombre con el que viaja el dato. Obligatorio si quieres que el dato llegue al servidor.
- placeholder: texto de ejemplo que se muestra dentro del campo vacío y desaparece al escribir. Es una pista visual, no un valor real.
- value: el valor inicial del campo. A diferencia del placeholder, el value sí es un dato real que se envía si el usuario no lo cambia. También sirve para ponerle texto a los botones.
- required: sin valor, solo se escribe la palabra. Hace que el campo sea obligatorio: el navegador no deja enviar el formulario si está vacío.
- disabled: desactiva el campo: se ve opaco y no se puede usar ni se envía su dato.
- readonly: el campo se puede leer pero no modificar. A diferencia de disabled, su dato sí se envía.
- maxlength: limita cuántos caracteres puede escribir el usuario.
- minlength: exige un mínimo de caracteres.

```
<input type='text' name='usuario' placeholder='Elige tu usuario' maxlength='20'
required>
<input type='text' name='codigo' value='PROM02026' readonly>
```

En el primer ejemplo el campo es obligatorio, acepta máximo 20 caracteres y muestra una pista. En el segundo el campo ya trae un valor que el usuario puede ver pero no cambiar.

Una confusión frecuente: placeholder no sustituye a label. El placeholder desaparece cuando escribes, así que si el usuario olvida qué pedía el campo, ya no tiene cómo saberlo. Además, los lectores de pantalla no siempre lo anuncian bien. La forma correcta es usar siempre un label visible y usar el placeholder solo como ayuda extra.

label: el nombre visible de cada campo

La etiqueta label le pone un nombre visible a cada campo del formulario. Pero su verdadero poder está en el atributo for: cuando el for del label coincide con el id del campo, quedan vinculados, y entonces hacer clic en el texto del label activa el campo automáticamente.

```
<label for='nombre'>Nombre completo:</label>
<input type='text' id='nombre' name='nombre'>
```

Haz la prueba: en un formulario bien hecho, si haces clic en las palabras 'Nombre completo:', el cursor salta solito al campo. Eso pasa gracias al vínculo entre for='nombre' e id='nombre'. Fíjate en algo importante: aquí el id sí tiene un trabajo que solo él puede hacer, porque el for necesita un identificador único al cual apuntar. Si dos campos tuvieran el mismo id, el label no sabría a cuál señalar.

¿Por qué importa tanto el label? Por tres razones. Primera, usabilidad: el área clicable se hace más grande, lo cual ayuda muchísimo en celulares, donde los campos pequeños son difíciles de tocar. Segunda, accesibilidad: los lectores de pantalla anuncian el label cuando el usuario llega al campo, así las personas con discapacidad visual saben qué deben escribir. Tercera, claridad: un formulario con labels se entiende a simple vista, sin adivinanzas.

El error clásico es escribir el texto del campo como texto suelto, sin label:

```
<!-- MAL: texto suelto, sin vínculo -->
Nombre: <input type='text' name='nombre'>

<!-- BIEN: label vinculado con for -->
<label for='nombre'>Nombre:</label>
<input type='text' id='nombre' name='nombre'>
```

La primera versión se ve igual, pero no tiene ninguno de los beneficios. Acostúmbrate desde hoy: cada campo de tus formularios lleva su label con for. Sin excepciones.

button: los botones de acción

La etiqueta button crea un botón. Su comportamiento depende del atributo type, que tiene tres valores:

- type='submit': envía el formulario. Es el botón de 'Enviar', 'Registrarme', 'Guardar'.
- type='reset': borra todo lo que el usuario escribió y deja el formulario como estaba al inicio.
- type='button': no hace nada por sí solo; sirve para que JavaScript le asigne una acción después.

```
<button type='submit'>Enviar mensaje</button>
<button type='reset'>Borrar todo</button>
<button type='button'>Mostrar ayuda</button>
```

Un detalle traicionero: si no escribes el type, el botón se comporta como submit por defecto. Eso significa que un botón que pusiste solo para 'mostrar ayuda' podría enviar el formulario sin querer si olvidaste ponerle type='button'. Este es un error que hasta programadores con experiencia cometen. Regla de oro: siempre escribes el type de tus botones, aunque parezca obvio.

También puedes crear botones con input type='submit', type='reset' y type='button', usando el atributo value para el texto. Pero la etiqueta button es más flexible porque dentro de ella puedes poner texto con formato, imágenes o íconos, mientras que input solo acepta texto plano.

textarea: para textos largos

Cuando necesitas que el usuario escriba más que unas palabras, como un mensaje, un comentario o una descripción, input type='text' se queda corto: es de una sola línea. Para eso existe textarea, que crea un cuadro de texto de varias líneas.

```
<label for='mensaje'>Tu mensaje:</label>
<textarea id='mensaje' name='mensaje' rows='5' cols='40' placeholder='Escribe tu mensaje aquí'></textarea>
```

A diferencia de input, textarea sí tiene etiqueta de cierre, y el texto inicial se escribe entre las etiquetas, no en un atributo value. Los atributos rows y cols controlan el tamaño visible: rows es el número de líneas y cols el ancho aproximado en caracteres. El usuario puede escribir todo lo que quiera aunque el cuadro se vea pequeño: aparecen barras de desplazamiento

automáticamente. Es el campo ideal para formularios de contacto, cajas de comentarios y descripciones de productos.

select y option: listas desplegables

Cuando quieres que el usuario elija una opción de una lista cerrada, como su país, su ciudad o su talla de playera, la herramienta perfecta es select con sus option. Es la clásica lista desplegable.

```
<label for='pais'>País:</label>
<select id='pais' name='pais'>
  <option value='mx'>México</option>
  <option value='ar'>Argentina</option>
  <option value='co'>Colombia</option>
  <option value='es'>España</option>
</select>
```

El usuario ve los nombres de los países, pero lo que viaja al servidor es el value: 'mx', 'ar', 'co' o 'es'. Esto es muy útil porque te permite mostrar textos amigables mientras guardas códigos cortos y limpios en tu base de datos.

Puedes marcar una opción como seleccionada desde el inicio con el atributo selected:

```
<select id='talla' name='talla'>
  <option value='ch'>Chica</option>
  <option value='m' selected>Mediana</option>
  <option value='g'>Grande</option>
</select>
```

También puedes agrupar opciones relacionadas con optgroup, que muestra un título de grupo no seleccionable:

```

<label for='curso'>Elige tu curso:</label>
<select id='curso' name='curso'>
  <optgroup label='Programación'>
    <option value='html5'>HTML5 desde cero</option>
    <option value='css'>CSS para principiantes</option>
    <option value='js'>JavaScript básico</option>
  </optgroup>
  <optgroup label='Diseño'>
    <option value='ux'>Diseño UX</option>
    <option value='figma'>Figma práctico</option>
  </optgroup>
</select>

```

Y si agregas el atributo `multiple` al `select`, el usuario puede elegir varias opciones a la vez manteniendo presionada la tecla `Ctrl` o `Cmd`. No lo uses sin avisarle al usuario cómo funciona, porque mucha gente no sabe que se puede.

Los inputs inteligentes de HTML5

Aquí está una de las mejores noticias de esta unidad: HTML5 trajo tipos de `input` que hacen trabajo por ti. Antes, validar que un correo estuviera bien escrito o mostrar un calendario requería programar mucho con JavaScript. Ahora el navegador lo hace solo, gratis. Son como asistentes que vienen incluidos con cada campo. Vamos a conocerlos todos.

`type='email'`: campo para correos electrónicos. El navegador verifica solo que lo escrito tenga forma de correo: algo, arroba, algo, punto, algo. Si el usuario escribe 'juan' sin arroba y trata de enviar, el navegador le muestra un aviso y no deja continuar.

```

<label for='correo'>Correo electrónico:</label>
<input type='email' id='correo' name='correo' placeholder='tucorreo@ejemplo.com'
required>

```

`type='password'`: campo para contraseñas. Lo que el usuario escribe se muestra como puntos o asteriscos, para que nadie lo lea por encima del hombro. Ojo: esto solo oculta el texto en pantalla; la protección real de la contraseña depende de que el formulario use `method='post'` y de que la página tenga conexión segura.

```

<label for='clave'>Contraseña:</label>
<input type='password' id='clave' name='clave' minlength='8' required>

```

`type='date'`: abre un calendario donde el usuario elige la fecha con clics, sin tener que adivinar el formato. El dato viaja en formato año-mes-día, siempre igual, lo cual te ahorra dolores de cabeza.

```
<label for='nacimiento'>Fecha de nacimiento:</label>
<input type='date' id='nacimiento' name='nacimiento'>
```

También existen `type='time'` para horas, `type='month'` para mes y año, y `type='week'` para semanas, aunque `date` es el más usado.

`type='color'`: abre un selector de color donde el usuario elige visualmente. Perfecto para personalizaciones, como elegir el color de una playera o de un perfil.

```
<label for='color-fav'>Tu color favorito:</label>
<input type='color' id='color-fav' name='color-fav' value='#3366ff'>
```

`type='number'`: campo para números, con flechitas para subir y bajar el valor. Acepta los atributos `min`, `max` y `step` para controlar el rango y los saltos.

```
<label for='edad'>Edad:</label>
<input type='number' id='edad' name='edad' min='1' max='120'>
<label for='cantidad'>Cantidad (de 2 en 2):</label>
<input type='number' id='cantidad' name='cantidad' min='0' max='100' step='2'>
```

Con `min='1'` `max='120'` el navegador no acepta edades imposibles. Con `step='2'` el valor sube de dos en dos: 0, 2, 4, 6. Es ideal para cantidades, edades, precios y cualquier número con límites.

`type='tel'`: campo para teléfonos. En computadoras se ve como un texto normal, pero en celulares hace algo genial: abre el teclado numérico en vez del teclado de letras, lo cual facilita muchísimo escribir el número. El navegador no valida el formato porque los teléfonos varían mucho entre países.

```
<label for='telefono'>Teléfono:</label>
<input type='tel' id='telefono' name='telefono' placeholder='55 1234 5678'>
```

`type='url'`: campo para direcciones web. El navegador verifica que lo escrito parezca una dirección válida, y en celulares muestra un teclado con teclas útiles como `'.com'` y la diagonal.

```
<label for='sitio'>Tu sitio web:</label>
<input type='url' id='sitio' name='sitio' placeholder='https://tusitio.com'>
```

type='search': campo de búsqueda. Se ve parecido al texto normal, pero en algunos navegadores muestra una equis para borrar lo escrito de un golpe, y en celulares abre el teclado con el botón de buscar.

```
<label for='buscar'>Buscar:</label>
<input type='search' id='buscar' name='q' placeholder='¿Qué buscas?'>
```

type='range': una barra deslizante para elegir un valor dentro de un rango. Es perfecta cuando el valor exacto no importa tanto, como el volumen, el nivel de satisfacción o un presupuesto aproximado. Usa min, max, step y value para configurarla.

```
<label for='satisfaccion'>Nivel de satisfacción:</label>
<input type='range' id='satisfaccion' name='satisfaccion' min='0' max='10'
value='5'>
```

type='checkbox': casilla de verificación que se puede activar o desactivar. Sirve para opciones independientes donde se puede elegir varias a la vez, como pasatiempos o para aceptar términos y condiciones. Con el atributo checked aparece ya marcada.

```
<p>¿Qué temas te interesan? (puedes elegir varios)</p>
<input type='checkbox' id='tema-html' name='temas' value='html' checked>
<label for='tema-html'>HTML</label>
<input type='checkbox' id='tema-css' name='temas' value='css'>
<label for='tema-css'>CSS</label>
<input type='checkbox' id='tema-js' name='temas' value='js'>
<label for='tema-js'>JavaScript</label>
```

type='radio': botón de opción circular donde solo se puede elegir UNA opción del grupo. El truco está en que todos los radio del mismo grupo comparten el mismo name: eso le dice al navegador que son excluyentes entre sí.

```
<p>¿Cuál es tu nivel?</p>
<input type='radio' id='nivel-basico' name='nivel' value='basico' checked>
<label for='nivel-basico'>Básico</label>
<input type='radio' id='nivel-intermedio' name='nivel' value='intermedio'>
<label for='nivel-intermedio'>Intermedio</label>
<input type='radio' id='nivel-avanzado' name='nivel' value='avanzado'>
<label for='nivel-avanzado'>Avanzado</label>
```

La diferencia entre checkbox y radio es una de las preguntas clásicas: checkbox es para 'elige todas las que apliquen' y radio es para 'elige solo una'. Si pones radios con name distintos, cada uno funcionaría por separado y el usuario podría marcarlos todos, rompiendo la lógica de 'solo una'. Así que recuerda: mismo grupo de radio, mismo name.

type='file': permite al usuario elegir un archivo de su dispositivo para subirlo, como una foto de perfil o un documento. El atributo accept limita qué tipos de archivo se pueden elegir.

```
<label for='foto'>Sube tu foto de perfil:</label>
<input type='file' id='foto' name='foto' accept='image/*'>
```

Con accept='image/*' el selector solo muestra imágenes. Importante: los formularios que suben archivos necesitan method='post' en el form; con get no funciona.

type='submit', type='reset' y type='button': son los botones creados con input en vez de con la etiqueta button. El texto del botón se define con value.

```
<input type='submit' value='Enviar formulario'>
<input type='reset' value='Limpiar'>
```

Un consejo práctico sobre compatibilidad: los navegadores viejos que no conocen un tipo nuevo de input lo tratan como type='text' normal. Eso significa que tu formulario sigue funcionando aunque el input inteligente no aparezca: el usuario simplemente escribe el dato a mano. Esta cualidad, llamada degradación elegante, hace que sea seguro usar los tipos de HTML5 sin miedo.

Errores comunes que debes evitar

Vamos a repasar los errores que más cometen los principiantes con atributos y formularios, para que tú no caigas en ellos.

Error 1: ids duplicados. Recuerda que el id es único por página. Si copias y pegas una sección y se te olvida cambiar el id, tendrás dos elementos con el mismo identificador. El navegador no te avisa, pero el label apuntará al primero que encuentre, los estilos se aplicarán de forma rara y JavaScript se confundirá. Cada vez que copies código con ids, cámbialos inmediatamente.

Error 2: label sin for. Escribir el texto del campo como texto suelto, sin la etiqueta label vinculada, es el error de accesibilidad más común. Se ve igual, pero pierdes el clic que activa el campo y los lectores de pantalla no saben qué pedir. Cada input, textarea y select merece su label con for apuntando al id del campo.

Error 3: olvidar el name en los inputs. Sin name, el dato del campo no viaja al servidor. Es un error silencioso y desesperante: todo se ve bien, el botón funciona, pero no llega nada. Revisa siempre que cada campo que deba enviarse tenga su name.

Error 4: button sin type. Un button sin type actúa como submit. Si lo pusiste para abrir una ayuda o limpiar algo con JavaScript y olvidaste el type, enviará el formulario por accidente. Escribe siempre el type: submit, reset o button.

Error 5: usar placeholder como si fuera label. El placeholder desaparece al escribir y no todos los lectores de pantalla lo anuncian. Úsalo como pista extra, nunca como el único nombre del campo.

Error 6: radios con name distintos cuando deberían ser excluyentes. Si cada radio tiene su propio name, el usuario puede marcarlos todos y se pierde la lógica de 'elige solo una'. Los radios del mismo grupo comparten el name.

Error 7: method='get' para datos delicados. Si tu formulario pide contraseñas o datos personales y usas get, esa información queda visible en la dirección web y en el historial. Para datos privados o acciones que cambian algo, usa post.

Error 8: poner atributos en la etiqueta de cierre. Los atributos van solo en la etiqueta de apertura. Parece obvio, pero cuando estás cansado es fácil equivocarse.

Buenas prácticas y accesibilidad básica en formularios

Un buen formulario no solo funciona: es claro, amable y usable por todas las personas, incluyendo quienes usan lectores de pantalla o navegan solo con teclado. Estas prácticas te separan de los principiantes.

Primera: un label visible para cada campo, siempre vinculado con for. Ya lo viste, pero vale repetirlo porque es la práctica más importante de todas.

Segunda: usa los tipos de input correctos. No uses `type='text'` para todo. Si pides un correo, usa `email`; si pides una fecha, usa `date`; si pides un teléfono, usa `tel`. Cada tipo correcto le regala al usuario validación gratis y el teclado adecuado en su celular. Es de los detalles que hacen que un formulario se sienta profesional.

Tercera: marca los campos obligatorios con `required` y dilo con palabras. Además del atributo `required`, escribe '(obligatorio)' en el label o marca con un asterisco explicado al inicio del formulario. No dejes que el usuario adivine.

Cuarta: agrupa campos relacionados con `fieldset` y `legend`. Si tu formulario tiene secciones, como 'Datos personales' y 'Datos de envío', envuelve cada grupo en `fieldset` y ponle un título con `legend`. Los lectores de pantalla anuncian estos grupos y el formulario se vuelve mucho más fácil de entender.

```
<fieldset>
  <legend>Datos personales</legend>
  <label for='nombre2'>Nombre:</label>
  <input type='text' id='nombre2' name='nombre2'>
  <label for='correo2'>Correo:</label>
  <input type='email' id='correo2' name='correo2'>
</fieldset>
```

Quinta: mensajes de error claros y amables. Cuando uses `required` o validaciones, el navegador muestra sus propios avisos. Más adelante con JavaScript podrás personalizarlos, pero desde hoy procura que tus labels y placeholders dejen clarísimo qué formato esperas, por ejemplo `placeholder='tucorreo@ejemplo.com'` en vez de `placeholder='correo'`.

Sexta: el orden lógico importa. Coloca los campos en el orden en que una persona los llenaría naturalmente: nombre, correo, mensaje. Y no pidas datos que no necesitas: cada campo extra es una razón más para abandonar el formulario. Los formularios cortos se completan más.

Séptima: el botón de envío debe decir qué hace. 'Enviar' está bien, pero 'Registrarme', 'Enviar mensaje' o 'Crear mi cuenta' es mejor: le dice al usuario exactamente qué va a pasar al presionarlo.

Con esto cierras la parte teórica. Ya sabes qué son los atributos, conoces los globales uno por uno, entiendes la diferencia entre `id` y `class`, y dominas los formularios con todos sus campos y tipos inteligentes.

Atributos y formularios trabajando juntos: anatomía de una página real

Hasta ahora viste atributos y formularios por separado, pero en una página real trabajan en equipo todo el tiempo. Vamos a ver una página completa de inscripción a un curso donde cada atributo cumple su función, como los músicos de una orquesta donde cada instrumento tiene su momento.

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Inscripción al curso de HTML5 - Lucas EGA Academy</title>
</head>
<body>

  <!-- Encabezado único de la página: id porque solo hay uno -->
  <header id='encabezado'>
    <h1 class='titulo' title='Academia de tecnología en español'>
      Lucas EGA Academy
    </h1>
    <p class='lema'>Aprende HTML5 desde cero</p>
  </header>

  <!-- Contenido principal -->
  <main id='contenido'>
    <h2>Inscríbete al curso</h2>
    <p>Completa el formulario y aparta tu lugar. Es gratis.</p>

    <!-- El formulario: post porque guarda datos del estudiante -->
    <form action='/inscribir' method='post' id='form-inscripcion'>

      <fieldset>
        <legend>Tus datos</legend>

        <label for='nombre-insc'>Nombre completo: (obligatorio)</label>
        <input type='text' id='nombre-insc' name='nombre'
          placeholder='Ej. Pedro Ramírez' required>

        <label for='correo-insc'>Correo electrónico: (obligatorio)</label>
        <input type='email' id='correo-insc' name='correo'
          placeholder='tucorreo@ejemplo.com' required>

        <label for='nacimiento-insc'>Fecha de nacimiento:</label>
        <input type='date' id='nacimiento-insc' name='nacimiento'>

      </fieldset>
    </form>
  </main>
</body>
</html>
```



```

</fieldset>

<fieldset>
  <legend>Tu nivel y tus intereses</legend>

  <p>¿Cuál es tu nivel actual? (elige uno)</p>
  <input type='radio' id='nv1' name='nivel' value='cero' checked>
  <label for='nv1'>Empiezo desde cero</label>
  <input type='radio' id='nv2' name='nivel' value='algo'>
  <label for='nv2'>Ya sé algo</label>

  <p>¿Qué quieres aprender después? (puedes elegir varios)</p>
  <input type='checkbox' id='fx-css' name='futuro' value='css'>
  <label for='fx-css'>CSS</label>
  <input type='checkbox' id='fx-js' name='futuro' value='js'>
  <label for='fx-js'>JavaScript</label>

  <label for='horario'>Horario preferido:</label>
  <select id='horario' name='horario'>
    <option value='manana'>Por la mañana</option>
    <option value='tarde' selected>Por la tarde</option>
    <option value='noche'>Por la noche</option>
  </select>
</fieldset>

<!-- Dato guardado en silencio para el sistema -->
<input type='hidden' name='origen' value='pagina-inscripcion'>

<button type='submit' class='boton-principal'>
  Apartar mi lugar gratis
</button>
</form>

<!-- Mensaje escondido: aparecerá al inscribirse -->
<p id='mensaje-exito' class='aviso' hidden>
  ¡Listo! Revisa tu correo para confirmar tu lugar.
</p>
</main>

<!-- Pie único de la página -->
<footer id='pie'>
  <p class='texto-pie'>Lucas EGA Academy - Aprende tecnología en español</p>
</footer>

```

```
</body>  
</html>
```

Recorre la página con ojos de detective y encuentra cada concepto de la unidad. El html lleva lang='es' para declarar el idioma. El header y el footer usan id porque son únicos; el main también. Los títulos y textos repetibles usan class. El formulario usa method='post' porque registra a una persona. Cada campo tiene su label con for vinculado al id. Los campos clave son required y lo dicen con palabras. El email se válida solo. Los radios comparten name='nivel' para elegir uno solo. Los checkbox permiten varios. El select trae una opción con selected. Hay un input type='hidden' que guarda el dato 'origen' sin mostrar nada en pantalla: viaja con el formulario pero el usuario nunca lo ve, útil para que el servidor sepa de qué página vino la inscripción. El botón lleva class='boton-principal' para darle estilo después y su texto dice exactamente qué hace. Y al final hay un mensaje de éxito con hidden, escondido hasta que JavaScript lo muestre cuando la inscripción se complete. Esta página usa casi todo lo que aprendiste en la unidad en un solo documento coherente. Si entiendes cada línea de este ejemplo, dominas la unidad.

Preguntas frecuentes de principiantes

¿El id y el name son lo mismo? No, y confundirlos es muy común. El id identifica al elemento dentro de la página: lo usan el label con for, CSS y JavaScript. El name es el nombre con el que viaja el dato del campo al servidor cuando se envía el formulario. Un input normalmente lleva los dos: id para vincularlo con su label y name para que su dato llegue al servidor. Pueden tener el mismo valor o distinto, no importa, mientras el id sea único en la página.

¿Puedo ponerle class a un solo elemento? Sí, sin problema. Una clase no está obligada a repetirse: es un grupo que por ahora tiene un solo miembro. De hecho es una buena costumbre usar class incluso para cosas únicas que podrían repetirse en el futuro, y reservar el id para lo que de verdad necesita ser único, como el ancla de un label o el objetivo de JavaScript.

¿Por qué mi placeholder no se envía como dato? Porque el placeholder no es un valor, es solo una pista visual. Si el usuario no escribe nada, el campo viaja vacío. Si quieres un valor inicial real que sí se envíe, usa el atributo value. Y recuerda: el placeholder no sustituye al label.

¿Debo validar en el servidor aunque HTML5 valide en el navegador? Sí, siempre. La validación de HTML5 con required, type='email' o min y max es una cortesía para el usuario: le avisa rápido y sin recargas. Pero un usuario avanzado puede saltársela, así que el programa del servidor siempre debe revisar los datos otra vez. Piensa en la validación del navegador como el filtro de la entrada y en la del servidor como la revisión final: necesitas las dos.

¿Qué pasa si el navegador del visitante es muy viejo y no conoce `type='date'`? Nada grave: lo trata como un campo de texto normal y el usuario escribe la fecha a mano. Tu formulario sigue funcionando. Por eso es seguro usar los tipos nuevos de HTML5: en el peor de los casos se degradan a texto.

¿Puedo tener varios formularios en una página? Sí, todos los que quieras. Cada form es independiente: tiene su propio action, su propio method y su propio botón de envío. Lo único que no puedes hacer es meter un form dentro de otro form: los formularios anidados no funcionan y el navegador se confunde.

¿Por qué los radio necesitan el mismo name y los checkbox no? Porque el name compartido es la señal que le dice al navegador 'estos son excluyentes, solo uno puede estar activo'. Los checkbox son independientes por naturaleza: cada uno vive su propia vida de activado o desactivado, así que no necesitan esa señal. Si les das el mismo name a varios checkbox, el servidor recibe varios valores con ese nombre, lo cual también es válido y útil.

¿El atributo hidden es seguro para ocultar información secreta? No. Hidden solo esconde el elemento de la vista, pero sigue en el código fuente y cualquiera puede verlo con las herramientas del navegador. Nunca escondas contraseñas, precios reales ni datos privados con hidden. Úsalo solo para contenido visual que se mostrará después, como mensajes o secciones plegables.

Plan de práctica de cinco días para dominar la unidad

Leer la teoría una vez no basta: los atributos y los formularios se aprenden con las manos en el teclado. Aquí tienes un plan de cinco días, con sesiones de unos 30 minutos, para que esta unidad se te quede grabada.

Día 1: atributos globales. Crea una página sencilla con un encabezado, tres párrafos y una imagen. Ponle id al encabezado, la misma class a los tres párrafos, title a la imagen y esconde uno de los párrafos con hidden. Ábrela en el navegador y juega: pasa el mouse para ver el title, quita el hidden y mira cómo aparece el párrafo. El objetivo es que tus dedos escriban id y class sin pensarlo.

Día 2: tu primer formulario. Copia el formulario de contacto del Ejemplo 1 y modifícalo: cambia el action, agrega un campo de asunto con un select de tres opciones y un checkbox para recibir copia del mensaje. Prueba enviarlo vacío y con un correo mal escrito para ver la validación del navegador en acción. Fíjate en la dirección web si usas get y compárala con post para entender la diferencia con tus propios ojos.

Día 3: el zoológico de inputs. Crea una página de pruebas con un formulario que tenga un input de cada tipo: text, email, password, date, color, number con min y max, tel, url, search, range, checkbox, radio, file y los tres botones. Ábrela en tu celular además de tu computadora: vas a descubrir cómo el teclado cambia según el tipo de campo. Esa experiencia vale más que mil explicaciones.

Día 4: accesibilidad. Toma cualquier formulario que hayas hecho antes sin labels y reescríbelo con las buenas prácticas: label con for para cada campo, required anunciado con palabras, fieldset y legend para agrupar. Después navega tu formulario usando solo la tecla Tab, sin mouse. Si puedes llenarlo completo solo con el teclado, lo hiciste bien.

Día 5: proyecto integrador. Construye la página de inscripción del curso que viste en la anatomía de la página real, pero con tus propios datos: tu academia, tus cursos, tus preguntas. Usa id para lo único, class para lo repetible, data-* para guardar precios o códigos, hidden para un mensaje de éxito y todos los tipos de input que necesites. Cuando termines, revísala con la lista de errores comunes de esta unidad: ids duplicados, labels sin for, campos sin name, botones sin type. Si pasa la revisión, esta unidad es tuya.

Ejemplos paso a paso

Ejemplo 1: formulario de contacto completo

Este es el formulario más pedido en el mundo real: el clásico 'contáctanos' que toda página necesita. Usa form con action y method post, labels vinculados, varios tipos de input y un textarea. Léelo con calma y fíjate cómo cada atributo tiene un propósito.

```

<!-- Formulario de contacto de Lucas EGA Academy -->
<form action='/enviar-contacto' method='post'>

    <!-- Campo de nombre: texto normal, obligatorio -->
    <label for='nombre'>Nombre completo: (obligatorio)</label>
    <input type='text' id='nombre' name='nombre'
        placeholder='Ej. María López' required>

    <!-- Campo de correo: type email valida solo el formato -->
    <label for='correo'>Correo electrónico: (obligatorio)</label>
    <input type='email' id='correo' name='correo'
        placeholder='tucorreo@ejemplo.com' required>

    <!-- Campo de teléfono: opcional, abre teclado numérico en celular -->
    <label for='telefono'>Teléfono: (opcional)</label>
    <input type='tel' id='telefono' name='telefono'
        placeholder='55 1234 5678'>

    <!-- Lista desplegable: el usuario elige el motivo -->
    <label for='motivo'>Motivo del mensaje:</label>
    <select id='motivo' name='motivo'>
        <option value='duda'>Tengo una duda</option>
        <option value='sugerencia' selected>Quiero hacer una sugerencia</option>
        <option value='problema'>Reportar un problema</option>
        <option value='otro'>Otro motivo</option>
    </select>

    <!-- Cuadro de texto largo para el mensaje -->
    <label for='mensaje'>Tu mensaje: (obligatorio)</label>
    <textarea id='mensaje' name='mensaje' rows='6' cols='40'
        placeholder='Cuéntanos en qué te podemos ayudar...'
        required></textarea>

    <!-- Botones: enviar y limpiar -->
    <button type='submit'>Enviar mensaje</button>
    <button type='reset'>Borrar todo</button>
</form>

```

Qué observar en este ejemplo: el form usa method='post' porque un mensaje de contacto es información privada. Cada campo tiene su label con for vinculado al id del campo. Los campos importantes llevan required y además el label dice '(obligatorio)' con palabras. El correo usa type='email' para que el navegador valide el formato solo. El select trae una opción marcada con selected como valor inicial. El textarea tiene rows y cols para su tamaño. Y los botones

llevan su type explícito para evitar sorpresas. Si pruebas este formulario e intentas enviarlo vacío o con un correo mal escrito, el navegador te lo impedirá con sus propios avisos: esa es la validación gratuita de HTML5 trabajando para ti.

Ejemplo 2: formulario de registro con varios tipos de input

Un registro de usuario es el segundo formulario más común, y es perfecto para practicar los inputs inteligentes: fecha de nacimiento con calendario, color favorito con selector, contraseña oculta y casilla de términos.

```
<!-- Registro de nuevo estudiante -->
<form action='/registrar' method='post'>

  <fieldset>
    <legend>Datos de la cuenta</legend>

    <!-- Usuario: texto con límites de longitud -->
    <label for='usuario'>Nombre de usuario: (obligatorio)</label>
    <input type='text' id='usuario' name='usuario'
      placeholder='Sin espacios, ej. ana_dev'
      minlength='4' maxlength='20' required>

    <!-- Correo con validación automática -->
    <label for='correo-reg'>Correo electrónico: (obligatorio)</label>
    <input type='email' id='correo-reg' name='correo-reg'
      placeholder='tucorreo@ejemplo.com' required>

    <!-- Contraseña: se oculta mientras se escribe -->
    <label for='clave'>Contraseña: (mínimo 8 caracteres)</label>
    <input type='password' id='clave' name='clave'
      minlength='8' required>
  </fieldset>

  <fieldset>
    <legend>Datos personales</legend>

    <!-- Fecha con calendario integrado -->
    <label for='nacimiento'>Fecha de nacimiento:</label>
    <input type='date' id='nacimiento' name='nacimiento'>

    <!-- Número con rango permitido -->
    <label for='edad'>Edad:</label>
    <input type='number' id='edad' name='edad' min='13' max='120'>
```

```

<!-- Selector visual de color -->
<label for='color-fav'>Color favorito para tu perfil:</label>
<input type='color' id='color-fav' name='color-fav' value='#3366ff'>

<!-- Nivel con barra deslizante -->
<label for='experiencia'>¿Cuánta experiencia tienes programando?</label>
<input type='range' id='experiencia' name='experiencia'
      min='0' max='10' value='0'>
</fieldset>

<!-- Casilla obligatoria de términos -->
<input type='checkbox' id='terminos' name='terminos' required>
<label for='terminos'>Acepto los términos y condiciones (obligatorio)</label>

<button type='submit'>Crear mi cuenta</button>
</form>

```

Qué observar: los fieldset con legend agrupan los campos en dos secciones claras, 'Datos de la cuenta' y 'Datos personales', lo cual ayuda a los lectores de pantalla y a la vista. El usuario tiene minlength y maxlength para controlar la longitud. La contraseña usa type='password' y exige mínimo 8 caracteres. La fecha abre calendario, el color abre selector, y la experiencia usa una barra deslizante con valor inicial 0. La casilla de términos es required: sin aceptarla no hay registro. Fíjate también en los ids: como en esta misma página del libro ya usamos 'correo' antes, aquí usamos 'correo-reg' para no duplicar ids. En una página real cada id debe ser único en todo el documento.

Ejemplo 3: encuesta con radio, checkbox y select

Las encuestas combinan los tres tipos de selección: radio para elegir una sola opción, checkbox para elegir varias y select para listas largas. Este ejemplo también muestra la diferencia entre name compartido y name individual.

```

<!-- Encuesta: ¿cómo aprendes mejor? -->
<form action='/guardar-encuesta' method='post'>

    <!-- RADIO: solo una opción, mismo name en todo el grupo -->
    <p>¿Cuántas horas estudias al día? (elige una)</p>
    <input type='radio' id='h1' name='horas' value='menos1'>
    <label for='h1'>Menos de 1 hora</label>
    <input type='radio' id='h2' name='horas' value='1a2' checked>
    <label for='h2'>De 1 a 2 horas</label>
    <input type='radio' id='h3' name='horas' value='mas2'>
    <label for='h3'>Más de 2 horas</label>

    <!-- CHECKBOX: varias opciones, cada una con su propio id -->
    <p>¿Qué temas te gustaría aprender? (puedes elegir varios)</p>
    <input type='checkbox' id='t-html' name='temas' value='html'>
    <label for='t-html'>HTML5</label>
    <input type='checkbox' id='t-css' name='temas' value='css'>
    <label for='t-css'>CSS</label>
    <input type='checkbox' id='t-js' name='temas' value='js'>
    <label for='t-js'>JavaScript</label>
    <input type='checkbox' id='t-py' name='temas' value='python'>
    <label for='t-py'>Python</label>

    <!-- SELECT con grupos de opciones -->
    <label for='dispositivo'>¿En qué dispositivo estudias más?</label>
    <select id='dispositivo' name='dispositivo'>
        <optgroup label='Móviles'>
            <option value='android'>Celular Android</option>
            <option value='iphone'>iPhone</option>
            <option value='tablet'>Tableta</option>
        </optgroup>
        <optgroup label='Computadoras'>
            <option value='laptop'>Laptop</option>
            <option value='pc'>Computadora de escritorio</option>
        </optgroup>
    </select>

    <!-- Satisfacción con barra deslizante -->
    <label for='recomienda'>¿Qué tanto recomendarías la academia? (0 a 10)</label>
    <input type='range' id='recomienda' name='recomienda' min='0' max='10' value='8'>

    <button type='submit'>Enviar encuesta</button>
</form>

```


Qué observar: los tres radio comparten `name='horas'`, por eso solo se puede elegir uno; además uno trae `checked` como opción inicial. Los checkbox comparten `name='temas'` pero cada uno tiene su propio `id` y su propio `value`, y el usuario puede marcar varios. El select usa `optgroup` para separar móviles de computadoras. La barra de rango mide la recomendación del 0 al 10 con valor inicial 8. Este ejemplo es un excelente molde: cambiando las preguntas tienes lista cualquier encuesta.

Ejemplo 4: tarjeta de usuario usando `id` y `class`

Ahora un ejemplo sin formulario, para practicar atributos globales: una tarjeta de perfil donde `id` identifica lo único y `class` agrupa lo repetible. Imagina una sección de 'estudiantes destacados' con varias tarjetas iguales.

```

<!-- Sección de estudiantes destacados -->
<section id='estudiantes-destacados'>
  <h2 class='titulo-seccion'
    title='Estudiantes con mejor promedio del mes'>
    Estudiantes destacados
  </h2>

  <!-- Primera tarjeta -->
  <article class='tarjeta-estudiante' data-promedio='9.8'>
    <img src='ana.jpg' alt='Foto de Ana'
      title='Ana: promedio 9.8'>
    <h3 class='nombre-estudiante'>Ana Martínez</h3>
    <p class='dato-estudiante'>Curso: HTML5 desde cero</p>
    <p class='dato-estudiante'>País: <span lang='es'>México</span></p>
    <button type='button' class='boton-perfil'>Ver perfil</button>
  </article>

  <!-- Segunda tarjeta: misma estructura, mismos class -->
  <article class='tarjeta-estudiante' data-promedio='9.6'>
    <img src='luis.jpg' alt='Foto de Luis'
      title='Luis: promedio 9.6'>
    <h3 class='nombre-estudiante'>Luis Hernández</h3>
    <p class='dato-estudiante'>Curso: CSS para principiantes</p>
    <p class='dato-estudiante'>País: <span lang='es'>Colombia</span></p>
    <button type='button' class='boton-perfil'>Ver perfil</button>
  </article>

  <!-- Tercera tarjeta escondida: se mostrará después -->
  <article class='tarjeta-estudiante' data-promedio='9.5' hidden>
    <img src='sofia.jpg' alt='Foto de Sofía'
      title='Sofía: promedio 9.5'>
    <h3 class='nombre-estudiante'>Sofía Rojas</h3>
    <p class='dato-estudiante'>Curso: JavaScript básico</p>
    <p class='dato-estudiante'>País: <span lang='es'>Argentina</span></p>
    <button type='button' class='boton-perfil'>Ver perfil</button>
  </article>
</section>

```

Qué observar: la section tiene id='estudiantes-destacados' porque es única en la página. Las tres tarjetas comparten class='tarjeta-estudiante' porque tienen la misma estructura: con una sola regla de CSS les darás el mismo diseño a las tres. Lo mismo pasa con 'nombre-estudiante', 'dato-estudiante' y 'boton-perfil'. Cada tarjeta guarda su promedio en data-promedio, un dato privado que JavaScript podría leer para ordenarlas de mayor a menor

sin mostrarlo en pantalla. El h2 usa title para dar información extra al pasar el mouse. La tercera tarjeta lleva hidden: existe en el código pero no se muestra, lista para aparecer cuando se presione un botón de 'ver más'. Los botones usan type='button' porque no envían ningún formulario: su acción la definirá JavaScript después. Este ejemplo resume toda la primera mitad de la unidad en un caso real.

Práctica

Ejercicio 1: identifica y corrige los atributos

Enunciado: el siguiente código tiene cuatro errores relacionados con atributos. Encuéntralos, explica por qué son errores y escribe la versión corregida.

```
<h1 id='titulo'>Mi blog</h1>
<h2 id='titulo'>Entradas recientes</h2>
<p class='resumen' title=Consejo del día>Aprende algo nuevo hoy.</p>
<img src='foto.jpg' alt='Mi foto' style='color: red'>
<button>Comprar</button>
```

Solución paso a paso:

Paso 1: detecta el id duplicado. El h1 y el h2 usan id='titulo'. Recuerda la regla de oro: un id solo puede existir una vez por página. El navegador no marca error, pero los estilos y JavaScript se comportarán de forma impredecible. Corrección: deja id='titulo' en el h1 y cambia el del h2 por id='subtitulo', o usa una clase si habrá más subtítulos.

Paso 2: detecta el valor sin comillas. En el párrafo, title=Consejo del día no lleva comillas, así que el navegador solo tomará 'Consejo' como valor y el resto lo interpretará mal. Corrección: title='Consejo del día', siempre con comillas.

Paso 3: detecta el style mal usado. La imagen lleva style='color: red', pero color no afecta a una imagen: la propiedad correcta para teñir imágenes sería otra, y además el style en línea debería reservarse para pruebas. No es un error que rompa la página, pero es código sin efecto. Corrección: quítalo o úsalo con una propiedad que sí aplique, por ejemplo style='width: 200px'.

Paso 4: detecta el button sin type. El botón no tiene type, así que si algún día este código queda dentro de un form, enviará el formulario por accidente. Corrección: escribe siempre el type, en este caso type='button'.

Código corregido:

```
<h1 id='titulo'>Mi blog</h1>
<h2 id='subtitulo'>Entradas recientes</h2>
<p class='resumen' title='Consejo del día'>Aprende algo nuevo hoy.</p>
<img src='foto.jpg' alt='Mi foto' style='width: 200px'>
<button type='button'>Comprar</button>
```

Moraleja del ejercicio: la mayoría de los errores con atributos son silenciosos, el navegador no te avisa. Por eso la revisión visual cuidadosa es una habilidad tan valiosa como escribir el código.

Ejercicio 2: arma un formulario de suscripción al boletín

Enunciado: crea un formulario para suscribirse al boletín de Lucas EGA Academy. Debe pedir nombre y correo (ambos obligatorios), incluir una casilla para aceptar recibir promociones (opcional, marcada por defecto) y un botón que diga 'Suscribirme'. El formulario debe enviar los datos por post a '/suscribir'.

Solución paso a paso:

Paso 1: abre el form con action y method. Como es un registro que guarda datos del usuario, usa method='post', nunca get.

Paso 2: agrega el campo de nombre con su label vinculado. Usa type='text', ponle id y name, y márcalo required. El label lleva for con el mismo valor que el id.

Paso 3: agrega el campo de correo con type='email' para que el navegador valide el formato automáticamente. También required, con su label vinculado y un placeholder de ejemplo.

Paso 4: agrega el checkbox de promociones. Como es opcional, no lleva required. Para que aparezca marcada por defecto, agrega el atributo checked. Recuerda que el checkbox necesita su id para vincularse con el label mediante for.

Paso 5: cierra con el botón de envío usando type='submit' y un texto claro que diga qué hace.

Código solución:

```
<form action='/suscribir' method='post'>
  <label for='nombre-boletin'>Nombre: (obligatorio)</label>
  <input type='text' id='nombre-boletin' name='nombre-boletin'
    placeholder='Tu nombre' required>

  <label for='correo-boletin'>Correo electrónico: (obligatorio)</label>
  <input type='email' id='correo-boletin' name='correo-boletin'
    placeholder='tucorreo@ejemplo.com' required>

  <input type='checkbox' id='promos' name='promos' checked>
  <label for='promos'>Quiero recibir promociones y novedades</label>

  <button type='submit'>Suscribirme</button>
</form>
```

Prueba mental: si intentas enviarlo vacío, el navegador te detiene por los required. Si escribes un correo sin arroba, el type='email' te detiene. Si todo está bien, los datos viajan por post a '/suscribir' con los nombres nombre-boletin, correo-boletin y promos. Fíjate que cada campo tiene name: sin eso, los datos no llegarían.

Ejercicio 3: formulario de pedido con números, fecha y lista desplegable

Enunciado: una tienda en línea necesita un formulario de pedido con: cantidad de playeras (número entre 1 y 10), talla (lista desplegable con Chica, Mediana y Grande, con Mediana seleccionada por defecto), fecha preferida de entrega (calendario), color de la playera (selector de color) y método de pago (radio con tres opciones: tarjeta, transferencia y efectivo, solo una elegible). Todo se envía por post a '/pedido'.

Solución paso a paso:

Paso 1: el form con action='/pedido' y method='post', porque es una compra: datos delicados y una acción que crea algo.

Paso 2: la cantidad con type='number', min='1', max='10' y value='1' como valor inicial. El navegador impedirá cantidades fuera del rango.

Paso 3: la talla con select y tres option. Marca la mediana con selected. Los value son códigos cortos ('ch', 'm', 'g') mientras el usuario ve los nombres completos.

Paso 4: la fecha de entrega con type='date', que abre el calendario solo.

Paso 5: el color con type='color' y un value inicial en formato hexadecimal.

Paso 6: el método de pago con tres radio que comparten el mismo name='pago' para que solo se pueda elegir uno. Cada uno con su propio id vinculado a su label, y uno marcado con checked por defecto.

Código solución:

```
<form action='/pedido' method='post'>
  <label for='cantidad'>Cantidad de playeras (1 a 10):</label>
  <input type='number' id='cantidad' name='cantidad'
    min='1' max='10' value='1' required>

  <label for='talla'>Talla:</label>
  <select id='talla' name='talla'>
    <option value='ch'>Chica</option>
    <option value='m' selected>Mediana</option>
    <option value='g'>Grande</option>
  </select>

  <label for='entrega'>Fecha preferida de entrega:</label>
  <input type='date' id='entrega' name='entrega'>

  <label for='color-playera'>Color de la playera:</label>
  <input type='color' id='color-playera' name='color-playera' value='#ff0000'>

  <p>Método de pago: (elige uno)</p>
  <input type='radio' id='pago-tarjeta' name='pago' value='tarjeta' checked>
  <label for='pago-tarjeta'>Tarjeta</label>
  <input type='radio' id='pago-transf' name='pago' value='transferencia'>
  <label for='pago-transf'>Transferencia</label>
  <input type='radio' id='pago-efectivo' name='pago' value='efectivo'>
  <label for='pago-efectivo'>Efectivo</label>

  <button type='submit'>Confirmar pedido</button>
</form>
```

Qué practicaste aquí: number con min y max, select con selected, date, color y el grupo de radios con name compartido. Si por error le pusieras un name distinto a cada radio, el usuario podría elegir los tres métodos a la vez y el pedido llegaría con datos contradictorios. El name compartido es lo que hace la magia de 'solo una opción'.

Ejercicio 4: mejora la accesibilidad de un formulario mal hecho

Enunciado: este formulario funciona, pero tiene problemas de accesibilidad y usabilidad. Identifícalos y reescríbelo aplicando buenas prácticas.

```
<form action='/login' method='get'>
  Usuario: <input type='text' name='u'>
  Clave: <input type='text' name='p'>
  <button>Entrar</button>
</form>
```

Solución paso a paso:

Paso 1: detecta el method incorrecto. Es un inicio de sesión con contraseña: los datos son privados, así que method='get' los expondría en la dirección web. Corrección: method='post'.

Paso 2: detecta el tipo de input incorrecto para la clave. Usa type='text', así la contraseña se ve en pantalla mientras se escribe. Corrección: type='password'.

Paso 3: detecta los textos sueltos sin label. 'Usuario:' y 'Clave:' son texto plano, sin vínculo con los campos: no hay clic que active el campo y los lectores de pantalla no anuncian qué pedir. Corrección: usar label con for apuntando al id de cada campo.

Paso 4: detecta el button sin type. Dentro de un form funciona como submit por casualidad, pero es mejor ser explícito. Corrección: type='submit'.

Paso 5: mejora el texto del botón. 'Entrar' está bien, pero 'Iniciar sesión' describe mejor la acción.

Paso 6: agrega required a ambos campos y un placeholder de ayuda en el usuario.

Código solución:

```
<form action='/login' method='post'>
  <label for='usuario-login'>Usuario: (obligatorio)</label>
  <input type='text' id='usuario-login' name='usuario-login'
    placeholder='Tu nombre de usuario' required>

  <label for='clave-login'>Contraseña: (obligatorio)</label>
  <input type='password' id='clave-login' name='clave-login' required>

  <button type='submit'>Iniciar sesión</button>
</form>
```

Compara las dos versiones: la segunda se ve casi igual, pero es privada (post), oculta la contraseña (password), es navegable con clic en los labels, anuncia cada campo a los lectores de pantalla y no deja enviar campos vacíos. Esos detalles invisibles son los que separan un formulario amateur de uno profesional.

Ejercicio 5: galería de productos con data-*, hidden y clases

Enunciado: crea una sección de productos donde cada producto sea un article con la misma estructura usando class, cada uno guarde su precio en un atributo data-*, uno de los productos esté escondido con hidden (porque está agotado) y la sección completa tenga un id único. Incluye al menos tres productos.

Solución paso a paso:

Paso 1: abre una section con un id único, por ejemplo id='catalogo'. Solo hay un catálogo en la página, así que id es lo correcto.

Paso 2: crea cada producto como un article con class='producto'. Los tres comparten la clase porque tienen la misma estructura: imagen, nombre, descripción y precio visible.

Paso 3: en cada article agrega data-precio con el precio numérico y data-id con un identificador del producto. Estos datos quedan guardados en el elemento, listos para que JavaScript los lea después, por ejemplo para calcular el total del carrito.

Paso 4: al producto agotado agrégale el atributo hidden para que no se muestre, pero siga existiendo en el código por si vuelve a haber existencias.

Paso 5: usa title en las imágenes para dar información extra al pasar el mouse, y class en los elementos internos repetidos (nombre, precio, botón) para que después puedas darles estilo de un solo golpe.

Código solución:


```

<section id='catalogo'>
  <h2 class='titulo-seccion'>Nuestros productos</h2>

  <article class='producto' data-id='p001' data-precio='299'>
    <img src='playera.jpg' alt='Playera negra'
      title='Playera 100 por ciento algodón'>
    <h3 class='nombre-producto'>Playera clásica negra</h3>
    <p class='precio-producto'>299 pesos</p>
    <button type='button' class='boton-comprar'>Agregar al carrito</button>
  </article>

  <article class='producto' data-id='p002' data-precio='349'>
    <img src='sudadera.jpg' alt='Sudadera gris'
      title='Sudadera con capucha, tallas CH a G'>
    <h3 class='nombre-producto'>Sudadera gris con capucha</h3>
    <p class='precio-producto'>349 pesos</p>
    <button type='button' class='boton-comprar'>Agregar al carrito</button>
  </article>

  <article class='producto' data-id='p003' data-precio='199' hidden>
    <img src='gorra.jpg' alt='Gorra roja'
      title='Gorra ajustable, actualmente agotada'>
    <h3 class='nombre-producto'>Gorra roja</h3>
    <p class='precio-producto'>199 pesos</p>
    <button type='button' class='boton-comprar'>Agregar al carrito</button>
  </article>
</section>

```

Qué practicaste: id único para la sección, class para los elementos repetidos, data-* para guardar el precio y el identificador de cada producto, hidden para el agotado, title para ayuda emergente y type='button' en los botones porque su acción la definirá JavaScript más adelante. Este patrón de tarjeta con clases compartidas y datos privados es exactamente el que usan las tiendas en línea reales. Cuando llegues a CSS, con unas pocas reglas podrás convertir estas tarjetas simples en una galería con diseño profesional.

Resumen de la unidad

- Los atributos son información extra que se escribe dentro de la etiqueta de apertura con la forma nombre='valor'; van entre comillas y nunca en la etiqueta de cierre.

- Los atributos globales funcionan en casi todas las etiquetas: id, class, title, style, lang, data-*, hidden y tabindex.
- El atributo id es único por página: sirve para identificar un elemento específico, para que CSS le dé estilo individual, para que JavaScript lo encuentre y para crear enlaces que saltan a esa sección.
- El atributo class agrupa varios elementos: muchos elementos pueden compartir la misma clase y un elemento puede tener varias clases; es la herramienta favorita para aplicar lo mismo a muchos elementos de un golpe.
- Regla de oro: si es un elemento único y específico usa id; si se repite o se puede repetir usa class. En la práctica usarás class mucho más que id.
- El atributo title muestra ayuda emergente al pasar el mouse; style aplica estilos en línea solo para pruebas rápidas; lang indica el idioma del contenido; data-* guarda datos privados con el nombre que tú inventes; hidden esconde un elemento sin borrarlo; tabindex controla la navegación con teclado.
- Un formulario empieza con la etiqueta form y necesita action (a dónde van los datos) y method (cómo viajan).
- Con method='get' los datos viajan visibles en la dirección web: solo para búsquedas y filtros. Con method='post' viajan escondidos: obligatorio para contraseñas, datos personales y cualquier acción que crea o cambia algo.
- La etiqueta input se transforma con su atributo type: text, email, password, date, color, number, tel, url, search, range, checkbox, radio, file, submit, reset y button.
- Los inputs de HTML5 hacen trabajo gratis: email válida correos, password oculta el texto, date abre calendario, color abre selector de color, number acepta min, max y step, y tel abre el teclado numérico en celulares.
- El atributo name es el nombre con el que viaja cada dato: sin name el campo no se envía, aunque se vea perfecto.
- Cada campo lleva su label vinculado con for apuntando al id del campo: hace clicable el texto, ayuda en celulares y es esencial para lectores de pantalla.
- Los botones button siempre llevan su type explícito: submit envía, reset limpia y button espera una acción de JavaScript; sin type se comportan como submit por defecto.
- textarea es para textos largos de varias líneas; select con option crea listas desplegables donde el usuario ve un texto amable y el servidor recibe un value corto.

- En los radio, todos los del mismo grupo comparten el mismo name para que solo se pueda elegir uno; en los checkbox cada uno es independiente y se pueden elegir varios.
- Errores que ya no vas a cometer: ids duplicados, labels sin for, inputs sin name, buttons sin type, placeholder usado como label, radios con name distintos y get para datos delicados.
- Buenas prácticas: un label visible por campo, el tipo de input correcto para cada dato, required anunciado con palabras, fieldset y legend para agrupar, pedir solo los datos necesarios y botones con textos que digan qué hacen.
- Lo que aprendiste aquí es el puente hacia lo que sigue: id y class son los ganchos que usará CSS para el diseño, y los formularios serán el campo de juego de JavaScript para validar y dar vida a tus páginas.

UNIDAD 5: Validación, semántica y SEO

Bienvenido a la unidad cinco del Curso Completo de HTML5 de Lucas EGA Academy. Hasta este punto del curso has construido las piezas fundamentales de cualquier página web: dominas la estructura básica de un documento HTML, sabes usar encabezados, párrafos, listas y enlaces, trabajas con imágenes y multimedia, y además levantaste formularios completos con inputs de todo tipo. Eso ya es un logro serio. Pero una página que funciona no es lo mismo que una página que funciona bien. Esta unidad es la que separa a un programador aficionado de un desarrollador profesional, porque aquí vas a aprender tres habilidades que casi nadie enseña en serio: la validación nativa de formularios, el HTML semántico y el SEO básico.

La validación nativa es una de esas maravillas silenciosas de HTML5. Antes de que existiera, si querías obligar a un usuario a llenar un campo obligatorio, o verificar que un correo tuviera formato válido, o pedir una contraseña con un mínimo de caracteres, tenías que escribir código JavaScript a mano: decenas de líneas, condicionales, mensajes de error personalizados, funciones que se rompían en navegadores distintos. Con HTML5 todo eso se resuelve con atributos simples que el navegador entiende y ejecuta por ti. Le pones `required` a un input y el navegador no deja enviar el formulario hasta que ese campo este lleno. Le pones `type="email"` y el navegador revisa el formato del correo. Le pones `minlength="8"` y el navegador exige ocho caracteres como mínimo. Sin una sola línea de JavaScript. En esta unidad vas a dominar `required`, `minlength` y `maxlength`, `min` y `max` para números, `pattern` para validar con expresiones regulares simples, los tipos especiales `email`, `url`, `number`, `tel` y `date`, los mensajes de error que el navegador muestra automáticamente, y el atributo `novalidate` para los casos donde quieres tomar el control tu mismo.

El HTML semántico es el segundo pilar de esta unidad. La palabra semántica suena complicada pero la idea es simple: usar las etiquetas correctas para describir lo que cada parte de tu página significa. No es lo mismo envolver el encabezado de tu sitio en un `div` genérico que usar la etiqueta `header`, porque `header` le dice al mundo: esto es la cabecera del sitio. No es lo mismo usar `main` para el contenido principal, `nav` para la navegación, `article` para un artículo independiente, `section` para una sección temática, `aside` para contenido complementario y `footer` para el pie de página. A los humanos les da igual como se llame la etiqueta, pero a Google le importa muchísimo, y a los lectores de pantalla que usan las personas con discapacidad visual les importa todavía más. Un sitio semántico se posiciona mejor en buscadores, es accesible por defecto y es mucho más fácil de mantener. Vas a aprender cada etiqueta semántica con ejemplos reales, vas a armar la estructura completa de

una página profesional y vas a conocer el error más común de todos: usar div para absolutamente todo.

El tercer pilar es el SEO básico, que va de la mano con las meta etiquetas del head. El head de tu documento es la parte invisible pero poderosa: ahí viven el meta charset="UTF-8" que permite que tus acentos y ñes se vean bien en cualquier dispositivo, el meta viewport que hace que tu página se adapte al celular, el meta description que es el texto que Google muestra en los resultados de búsqueda, el title que es lo que aparece en la pestaña del navegador y en el título azul de Google, las etiquetas Open Graph que controlan como se ve tu página cuando alguien la comparte en redes sociales, y la jerarquía correcta de encabezados h1 a h6 que Google usa para entender de que trata tu página. En esta unidad vas a armar un head completo y optimizado, entender que son las keywords y por que ya no sirven, y aplicar todo en una landing page semántica de verdad.

Como encaja esta unidad en el curso: las unidades uno a cuatro te dieron las manos y los pies para caminar, esta unidad te da los ojos y la brújula. A partir de aquí, cada página que construyas va a validarse sola, va a hablar el idioma que Google entiende y va a verse bien en cualquier pantalla. Y lo mejor es que todo lo que aprendas aquí no requiere librerías, frameworks ni dependencias: es HTML puro, el estándar abierto de la web. Vamos a empezar.

Explicación detallada

La validación nativa: tu primera línea de defensa

Imagina que eres el portero de un antro. Tu trabajo es revisar que cada persona que entra cumpla las reglas: identificación en mano, edad mínima, vestimenta adecuada. Si alguien no cumple, no pasa. La validación de formularios funciona exactamente igual: es el filtro de seguridad que revisa los datos del usuario antes de que lleguen al servidor. Y aquí viene lo poderoso: el navegador puede ser ese portero sin que tu escribas ni una línea de código extra.

Antes de HTML5, la validación era un dolor de cabeza. Tenías que escribir funciones en JavaScript que revisaran campo por campo, mostrar mensajes de error con código, manejar casos especiales por navegador, y si JavaScript estaba desactivado, tu validación simplemente no existía. HTML5 cambió el juego por completo: el navegador trae el validador integrado, funciona igual en todos los dispositivos, muestra mensajes de error en el idioma del usuario y no se puede desactivar tan fácil. Tu solo declaras las reglas con atributos y el navegador las ejecuta.

La analogía del filtro de seguridad funciona en tres niveles. El primer nivel es el filtro grueso: impedir que entren datos vacíos o incompletos. El segundo nivel es el filtro fino: verificar formatos específicos como correos electrónicos, direcciones web y números dentro de rangos. El tercer nivel es el filtro personalizado: reglas a tu medida con expresiones regulares. Los tres niveles viven en atributos HTML que vas a dominar en esta unidad.

Un detalle importante que debes entender desde ya: la validación nativa del navegador es tu primera línea de defensa, pero no la única. Un usuario malintencionado puede desactivar JavaScript, modificar tu HTML o enviar datos directamente al servidor sin pasar por tu formulario. Por eso en proyectos profesionales siempre se valida también en el servidor. Pero eso no le quita mérito a la validación nativa: mejora la experiencia del usuario de forma inmediata, evita el 99 por ciento de los errores honestos y te ahorra cientos de líneas de código. Es gratis, es rápida y es estándar.

El atributo required: el campo obligatorio

El atributo required es el más simple y el más usado de todos. Se lo pones a un input, un textarea o un select, y el navegador se encarga de todo lo demás. Si el usuario intenta enviar el formulario con ese campo vacío, el navegador bloquea el envío, resalta el campo con un borde rojo y muestra un mensaje como "Completa este campo". Sin código, sin funciones, sin complicaciones.

Mira este ejemplo básico:

```
<form action="/registro" method="post">
  <label for="nombre">Nombre completo:</label>
  <input type="text" id="nombre" name="nombre" required>
  <button type="submit">Registrarse</button>
</form>
```

Con solo agregar la palabra required, ese campo se volvió obligatorio. Fíjate que required es un atributo booleano: no necesita valor, su sola presencia activa la regla. También puedes escribir required="required", pero la forma corta es la estándar y la que vas a ver en código profesional.

El comportamiento del navegador es inteligente: no solo revisa que el campo no esté vacío, también ignora los espacios en blanco. Si el usuario escribe solo espacios, el navegador lo considera vacío y bloquea el envío. Además, el mensaje de error aparece en el idioma del navegador del usuario: en español dirá algo como "Rellena este campo", en inglés "Please fill out this field". Tú no tienes que traducir nada.

Un error común de principiantes es poner required en el boton de envio o en elementos que no son campos de formulario. required solo funciona en input (excepto los tipos hidden, image, button, submit y reset), textarea y select. Si lo pones en un div o en un párrafo, no pasa nada, el navegador lo ignora silenciosamente.

Otro error frecuente es marcar TODOS los campos como obligatorios. Piensa en el usuario: si le pides su segundo apellido, su telefono y su fecha de nacimiento como obligatorios en un simple formulario de contacto, muchos van a abandonar el formulario. Marca como required solo lo que realmente necesitas. La validación también es diseño de experiencia de usuario.

minlength y maxlength: controlando el tamaño del texto

Los atributos minlength y maxlength controlan cuantos caracteres puede tener el valor de un campo de texto. minlength establece el mínimo y maxlength establece el máximo. Son perfectos para contraseñas, nombres de usuario, comentarios y cualquier texto con limites.

```
<label for="password">Contraseña (mínimo 8 caracteres):</label>
<input type="password" id="password" name="password" minlength="8" required>

<label for="usuario">Nombre de usuario (máximo 15 caracteres):</label>
<input type="text" id="usuario" name="usuario" maxlength="15" required>
```

En el primer ejemplo, si el usuario escribe una contraseña de 5 caracteres e intenta enviar, el navegador bloquea el envío y muestra algo como "Utiliza al menos 8 caracteres". En el segundo ejemplo, el navegador ni siquiera deja escribir el carácter número 16: el límite se aplica mientras el usuario teclea, en tiempo real.

Hay una diferencia sutil pero importante entre los dos: minlength se válida al momento de enviar el formulario, mientras que maxlength se aplica durante la escritura. Esto significa que maxlength es una restricción física (no puedes pasarte) y minlength es una restricción lógica (te avisan si no llegas).

Un caso de uso clásico: un campo de código postal mexicano tiene 5 dígitos. Puedes combinar ambos atributos para exigir exactamente 5 caracteres:

```
<label for="cp">Codigo postal:</label>
<input type="text" id="cp" name="cp" minlength="5" maxlength="5" required>
```

Pero ojo: minlength y maxlength cuentan caracteres, no verifican que sean números. El usuario podría escribir "abcde" y pasaría la validación de longitud. Para exigir que sean solo

números usarias `input type="number"` o el atributo `pattern`, que veremos más adelante. Cada atributo válida una cosa distinta y se pueden combinar para crear reglas más estrictas.

Error común: pensar que `maxlength` protege tu base de datos. No es así. `maxlength` solo limita lo que se escribe en el navegador, pero un atacante puede enviar datos más largos directamente al servidor. Siempre válida la longitud también del lado del servidor en aplicaciones reales.

min, max y step: rangos para números y fechas

Para los campos numéricos y de fecha, HTML5 ofrece `min`, `max` y `step`. Funcionan de forma parecida a `minlength` y `maxlength`, pero para valores numéricos en lugar de longitud de texto.

```
<label for="edad">Edad (entre 18 y 99):</label>
<input type="number" id="edad" name="edad" min="18" max="99" required>

<label for="cantidad">Cantidad (múltiplos de 5):</label>
<input type="number" id="cantidad" name="cantidad" min="5" max="100" step="5">

<label for="fecha">Fecha de cita (solo 2026):</label>
<input type="date" id="fecha" name="fecha" min="2026-01-01" max="2026-12-31"
required>
```

El atributo `step` define el incremento permitido: con `step="5"`, los valores válidos son 5, 10, 15, 20 y así sucesivamente. Si el usuario escribe 7, el navegador lo rechaza. Con fechas, `step` se mide en días: `step="7"` permitiría solo fechas separadas por semanas.

En los inputs de tipo `number`, el navegador muestra unas flechitas para subir y bajar el valor, y esas flechitas respetan automáticamente `min`, `max` y `step`. Es un detalle de experiencia de usuario que obtienes gratis.

Un truco muy usado: combinar `min` con la fecha de hoy para impedir fechas pasadas. Por ejemplo, en un formulario de reservación no quieres que el usuario elija ayer. Con un poco de JavaScript puedes poner el `min` dinámicamente, pero incluso el `min` fijo ya resuelve muchos casos.

Error común: usar `min` y `max` en un `input type="text"`. No funcionan ahí. `min` y `max` solo aplican a los tipos `number`, `range`, `date`, `month`, `week`, `time`, `datetime-local`. Si tu input es texto, usa `minlength` y `maxlength`, o `pattern`.

Los tipos especiales: email, url, number y tel

HTML5 agrega tipos de input que validan formatos automáticamente. Ya los viste en la unidad de formularios, pero ahora vas a entender su poder de validación a fondo.

El tipo email verifica que el valor tenga formato de correo electrónico: algo, arroba, algo, punto, algo. No verifica que el correo exista de verdad (eso es imposible desde el navegador), pero si atrapa errores de dedo como "juan@gmail" sin el.com, o "juan gmail.com" sin la arroba.

```
<label for="correo">Correo electronico:</label>
<input type="email" id="correo" name="correo" required>
```

El tipo url exige una dirección web completa, incluyendo el protocolo http:// o https://. Si el usuario escribe "google.com" sin el https://, el navegador lo rechaza. Esto confunde a muchos usuarios, así que en formularios reales a veces se agrega una ayuda visual o se usa pattern para hacerlo más flexible.

El tipo number solo acepta valores numericos. Si el usuario escribe letras, el navegador lo marca como invalido. Además en celulares muestra el teclado numerico, lo cual es una gran mejora de usabilidad en dispositivos móviles.

El tipo tel es un caso curioso: no válida nada por si mismo. Acepta cualquier texto. Su valor real es que en celulares muestra el teclado telefonico, con números grandes y fáciles de tocar. Para validar un formato de telefono específico, lo combinas con pattern.

```
<label for="telefono">Telefono (10 digitos):</label>
<input type="tel" id="telefono" name="telefono" pattern="[0-9]{10}" required>
```

Ese pattern="[0-9]{10}" significa: exactamente 10 digitos del 0 al 9. En la siguiente sección vas a dominar pattern a fondo.

El atributo pattern: validación con expresiones regulares

El atributo pattern es la herramienta más poderosa de la validación nativa. Te permite definir una regla personalizada usando una expresión regular, que es un mini lenguaje para describir formatos de texto. Suena intimidante, pero con los ejemplos de esta sección vas a manejar los casos más comunes sin problema.

Una expresión regular es como una plantilla: describes que caracteres estan permitidos y en que orden, y el navegador verifica que el valor del campo encaje en esa plantilla. Vamos con ejemplos de menor a mayor complejidad.

Solo letras (para nombres):

```
<input type="text" name="nombre" pattern="[A-Za-z]+" required>
```

La expresión `[A-Za-z]+` significa: una o más letras, mayusculas o minusculas. El signo más (+) quiere decir "uno o más". Si el usuario escribe "Juan123", no pasa la validación porque contiene números.

Solo letras incluyendo acentos y enes (importante en Mexico):

```
<input type="text" name="nombre" pattern="[A-Za-zÁÉÍÓÚŨÑáéíóúü]+" required>
```

Este es mejor para nombres reales como "Jose Maria" o "Penelope". Fijate que incluimos las vocales con tilde y la ene, porque pattern por defecto no las acepta en el rango A-Za-z.

Exactamente 10 digitos (telefono mexicano):

```
<input type="tel" name="telefono" pattern="[0-9]{10}" required>
```

Las llaves {10} significan "exactamente 10 veces". [0-9] significa "cualquier digito". Entonces `[0-9]{10}` es "exactamente 10 digitos".

Código postal mexicano (5 digitos):

```
<input type="text" name="cp" pattern="[0-9]{5}" required>
```

CURP simplificada (4 letras, 6 números, resto):

```
<input type="text" name="curp" pattern="[A-Z]{4}[0-9]{6}[A-Z0-9]{8}" required>
```

Esta es una versión simplificada: 4 letras mayusculas, 6 digitos de fecha, y 8 caracteres alfanumericos. La CURP real tiene reglas más específicas, pero este ejemplo te muestra como encadenar bloques en una expresión.

Contraseña segura (mínimo 8 caracteres, al menos una letra y un número):

```
<input type="password" name="password" pattern="(?=.*[A-Za-z])(?=.*[0-9]).{8,}" required>
```

Esta es más avanzada: `(?=.*[A-Za-z])` exige al menos una letra, `(?=.*[0-9])` exige al menos un número, y `.{8,}` exige mínimo 8 caracteres de cualquier tipo. No te preocupes si no la entiendes al cien por ciento todavía; las expresiones regulares son un tema enorme por si mismo. Lo importante es que sepas que `pattern` existe y que puedes buscar o pedir expresiones listas para los formatos comunes.

Puedes combinar `pattern` con el atributo `title` para dar una pista al usuario sobre el formato esperado. Cuando la validación falla, algunos navegadores muestran el contenido de `title` como ayuda:

```
<input type="tel" name="telefono" pattern="[0-9]{10}" title="Escribe 10 digitos sin espacios ni guiones" required>
```

Errores comunes con `pattern`: primero, olvidar que `pattern` se aplica al valor completo por defecto en HTML5 (no necesitas los símbolos de inicio y fin). Segundo, usar expresiones demasiado estrictas que rechazan datos validos, como un `pattern` de correo que no acepta correos con guiones. Tercero, no probar con datos reales: siempre prueba tu `pattern` con ejemplos validos e invalidos antes de publicar.

Mensajes de error del navegador: cómo funcionan

Cuando la validación falla, el navegador muestra un mensaje de error automáticamente en una burbuja junto al campo. Ese mensaje depende del tipo de error y del idioma del navegador. Estos son los mensajes típicos en español:

- Campo required vacío: "Rellena este campo."
- Email con formato invalido: "Incluye un signo "@" en la dirección de correo electrónico."
- Texto más corto que minlength: "Utiliza al menos X caracteres."
- Número fuera de rango: "El valor debe ser mayor o igual que X."
- `pattern` no cumplido: "El valor debe coincidir con el formato solicitado."

Además del mensaje, el navegador aplica estilos visuales automaticos: el campo invalido suele mostrar un borde rojo y el foco salta al primer campo con error. Todo esto ocurre sin que escribas código.

Puedes personalizar la apariencia de los campos validos e invalidos con CSS usando las pseudo clases:valid,:invalid,:required y:optional. Aunque CSS a fondo es tema de otro curso, vale la pena que conozcas estas porque trabajan directamente con la validación:

```
input:invalid {  
  border: 2px solid red;  
}  
input:valid {  
  border: 2px solid green;  
}
```

Con esas dos reglas, tus campos se pintan de rojo cuando el dato es invalido y de verde cuando es válido, en tiempo real, mientras el usuario escribe. Es una de las combinaciones más satisfactorias de HTML y CSS.

Un detalle fino::invalid se aplica también cuando el campo esta vacío y es required, incluso antes de que el usuario escriba nada. Esto puede hacer que un formulario recién cargado se vea todo rojo, lo cual asusta. La solución elegante es combinar con:not(:placeholder-shown) o aplicar los estilos solo después del primer intento de envío, pero eso ya entra en técnicas intermedias de CSS.

El atributo novalidate: cuando quieres tomar el control

Hay casos donde no quieres la validación automática del navegador. Por ejemplo, un formulario de "guardar borrador" donde el usuario puede guardar datos incompletos para continuar después, o un formulario donde vas a validar todo con tu propio JavaScript personalizado. Para esos casos existe el atributo novalidate, que se pone en la etiqueta form:

```
<form action="/borrador" method="post" novalidate>  
  <input type="email" name="correo">  
  <button type="submit">Guardar borrador</button>  
</form>
```

Con novalidate, el navegador envía el formulario sin revisar nada: ni required, ni pattern, ni tipos. Los atributos siguen ahí en el HTML, simplemente el navegador los ignora durante el envío.

También existe formnovalidate, que se pone en un botón de envío específico. Sirve cuando un formulario tiene dos botones: uno que válida y otro que no:

```
<form action="/procesar" method="post">
  <input type="email" name="correo" required>
  <button type="submit">Enviar (valida)</button>
  <button type="submit" formnovalidate>Guardar borrador (no valida)</button>
</form>
```

El primer boton válida normalmente, el segundo envía sin validar. Es una tecnica muy usada en formularios largos con opción de borrador.

Advertencia importante: usar novalidate no significa que los datos lleguen bien al servidor. Solo significa que el navegador no los revisa. Si desactivas la validación del navegador, la responsabilidad de validar recae en ti: en tu JavaScript y, sobre todo, en tu servidor. Nunca confies ciegamente en los datos que llegan de un formulario.

Qué significa semántica y por qué importa

La palabra semantica viene del griego semantikos, que significa "significativo". En el contexto de HTML, escribir código semantico significa usar etiquetas que describen el significado del contenido, no solo su apariencia. Es la diferencia entre decirle al navegador "aquí hay una caja" (div) y decirle "aquí hay el encabezado principal del sitio" (header), "aquí esta la navegación" (nav), "aquí esta el contenido principal" (main).

Piensa en la analogía del esqueleto etiquetado. Un esqueleto humano tiene huesos, y un medico puede nombrar cada uno: femur, tibia, craneo, costillas. Si en lugar de eso todos los huesos vinieran en una bolsa etiquetada como "huesos varios", el medico tendría que adivinar cual es cual. Un documento HTML hecho solo de divs es esa bolsa de huesos: el navegador ve cajas genericas y tiene que adivinar que es cada cosa. Un documento semantico es el esqueleto etiquetado: cada parte dice claramente que es.

A los usuarios humanos les da igual: un div con estilos se ve identico a un header con los mismos estilos. Pero hay dos lectores no humanos para los que la semantica lo es todo. El primero es Google: su robot rastrea tu página y usa las etiquetas semanticas para entender su estructura, decidir de que trata cada sección y posicionarte mejor en los resultados. El segundo son los lectores de pantalla: programas que usan las personas con discapacidad visual para navegar la web escuchando el contenido. Un lector de pantalla puede decir "navegación principal, 5 elementos" o "contenido principal" y saltar directamente ahi, pero solo si usaste nav y main. Con divs genericos, el usuario ciego tiene que escuchar toda la página de principio a fin.

La semántica también te ayuda a ti como desarrollador. Seis meses después de escribir una página, cuando vuelvas a su código, vas a entender mucho más rápido un documento con header, nav, main y footer que uno con veinte divs anidados con clases inventadas. El código semántico se documenta solo.

Las etiquetas semánticas una por una

Vamos a conocer cada etiqueta semántica principal con su propósito exacto y ejemplos de uso correcto.

header: representa el encabezado de una página o de una sección. Normalmente contiene el logotipo, el título del sitio y a veces la navegación principal. Puedes tener varios header en una página: uno para el sitio completo y otros para artículos individuales.

```
<header>
  <h1>Lucas EGA Academy</h1>
  <p>Aprende tecnología en español, gratis</p>
</header>
```

nav: envuelve los bloques de navegación principal: menús, listas de enlaces a las secciones del sitio. No todos los grupos de enlaces necesitan nav, solo la navegación principal. Los enlaces del pie de página, por ejemplo, normalmente no van en nav.

```
<nav>
  <ul>
    <li><a href="/cursos">Cursos</a></li>
    <li><a href="/blog">Blog</a></li>
    <li><a href="/contacto">Contacto</a></li>
  </ul>
</nav>
```

main: el contenido principal y único de la página. Solo debe haber un main por documento. Todo lo que se repite en varias páginas (cabecera, menú, pie) queda fuera de main. Los lectores de pantalla usan main para saltar directo al contenido.

```
<main>
  <h2>Bienvenido al curso de HTML5</h2>
  <p>Aquí empieza tu camino como desarrollador web...</p>
</main>
```

section: una sección temática del documento, un agrupamiento de contenido relacionado. Cada section debería tener idealmente su propio encabezado (h2, h3). Piensa en los capítulos de un libro: cada capítulo es una sección.

```
<section>
  <h2>Nuestros cursos</h2>
  <p>Tenemos 22 cursos gratuitos...</p>
</section>
```

article: contenido independiente y autocontenido que tendría sentido por si solo: un artículo de blog, una noticia, una publicación de foro, una tarjeta de producto. La prueba del article: si lo sacas de la página y lo pones en otro sitio, sigue teniendo sentido completo.

```
<article>
  <h2>Como validar formularios sin JavaScript</h2>
  <p>HTML5 incluye validacion nativa...</p>
</article>
```

aside: contenido complementario o tangencial: barras laterales, biografías del autor, enlaces relacionados, publicidad. Lo que va en aside debe estar relacionado con el contenido cercano, pero no es parte esencial de el. Si quitas el aside, el contenido principal sigue entendiéndose.

```
<aside>
  <h3>Articulos relacionados</h3>
  <ul>
    <li><a href="#">Guia de CSS</a></li>
    <li><a href="#">JavaScript basico</a></li>
  </ul>
</aside>
```

footer: el pie de página o de una sección. Normalmente contiene información de derechos de autor, enlaces secundarios, datos de contacto y avisos legales. Como header, puede haber varios footer en una página.

```
<footer>
  <p>2026 Lucas EGA Academy. Todos los derechos reservados.</p>
</footer>
```

Hay más etiquetas semanticas útiles: figure y figcaption para imágenes con pie de foto, time para fechas y horas (con el atributo datetime para que las máquinas las entiendan), mark para

resaltar texto, details y summary para contenido plegable nativo sin JavaScript. Las veras aplicadas en los ejemplos de esta unidad.

Estructura completa de una página semántica

Vamos a armar el esqueleto de una página real combinando todas las etiquetas. Esta es la estructura que Google ama y que los lectores de pantalla navegan con facilidad:

```
<body>
  <header>
    <h1>Mi sitio web</h1>
    <nav>
      <ul>
        <li><a href="/">Inicio</a></li>
        <li><a href="/cursos">Cursos</a></li>
        <li><a href="/contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <section>
      <h2>Ultimos articulos</h2>
      <article>
        <h3>Titulo del articulo</h3>
        <p>Contenido del articulo...</p>
      </article>
    </section>
    <aside>
      <h2>Enlaces de interes</h2>
      <p>Contenido complementario...</p>
    </aside>
  </main>
  <footer>
    <p>Derechos reservados 2026</p>
  </footer>
</body>
```

Fijate en el orden lógico: header con la identidad y la navegación, main con el contenido único (secciones, artículos y el aside complementario), y footer con el cierre. Esta estructura es tan estandar que los desarrolladores profesionales la reconocen al instante.

Una pregunta frecuente: article dentro de section o section dentro de article. Ambas son validas segun el caso. Si tienes una sección de "Últimas noticias" que agrupa varios artículos independientes, los article van dentro de la section. Si tienes un artículo largo dividido en capítulos, las section van dentro del article. La regla de oro: section agrupa contenido relacionado, article es contenido independiente.

El error de usar div para todo

El error más común en HTML es el "divitis": usar div para absolutamente todo. Un documento con divitis se ve así:

```
<div id="cabecera">
  <div class="menu">
    <div class="opcion"><a href="/">Inicio</a></div>
  </div>
</div>
<div id="contenido">
  <div class="nota">
    <div class="titulo">Mi articulo</div>
    <div class="texto">Contenido...</div>
  </div>
</div>
<div id="pie">Derechos reservados</div>
```

Funciona visualmente, pero es código mudo: ninguna etiqueta dice que significa. Para Google es una sopa de cajas sin jerarquía. Para un lector de pantalla es un laberinto sin señales. Para otro programador (o para ti dentro de seis meses) es un rompecabezas de clases inventadas.

La versión semantica del mismo código:

```

<header>
  <nav>
    <a href="/">Inicio</a>
  </nav>
</header>
<main>
  <article>
    <h2>Mi articulo</h2>
    <p>Contenido...</p>
  </article>
</main>
<footer>Derechos reservados</footer>

```

Menos código, más significado, mejor posicionamiento, mejor accesibilidad. Cuando tengas la tentación de escribir otro div, pregúntate: esto es un encabezado, una navegación, contenido principal, una sección, un artículo, un complemento o un pie. Si la respuesta es sí a alguna, usa la etiqueta semántica. Reserva el div para cuando de verdad necesites una caja genérica sin significado, por ejemplo para agrupar elementos solo con fines de diseño.

Ojo: ser semántico no significa prohibir los div. El div sigue siendo válido y necesario para envoltorios de diseño. El problema no es usar div, es usar SOLO div. La meta es un equilibrio: semántica donde hay significado, div donde solo hay presentación.

Meta charset UTF-8: tus acentos y ñes a salvo

El meta charset declara la codificación de caracteres del documento. La codificación es el diccionario que el navegador usa para convertir los bytes del archivo en letras visibles. Si el diccionario es incorrecto, los caracteres especiales se rompen y ves cosas como "JosÃ©" en lugar de "Jose" o "niÃ±o" en lugar de "niño". El estándar moderno es UTF-8, que cubre prácticamente todos los caracteres de todos los idiomas, incluyendo acentos, ñes, diéresis y emojis.

```
<meta charset="UTF-8">
```

Esta etiqueta va en el head, lo más arriba posible, idealmente como primera etiqueta después de abrir head. El navegador necesita saber la codificación antes de leer el resto del documento, así que mientras más arriba este, mejor.

Sin UTF-8, cada vez que escribas "información", "validación" o "Página" corres el riesgo de que se vean mal en algún dispositivo. Con UTF-8, tus textos en español se ven perfectos en

Mexico, Espana, Argentina y cualquier parte del mundo. Es una sola línea que te ahorra dolores de cabeza infinitos.

Además, tu archivo debe estar guardado realmente en codificación UTF-8. La mayoría de los editores modernos (VS Code, Sublime Text) guardan en UTF-8 por defecto, pero si usas el Bloc de notas de Windows, verifica que al guardar elijas UTF-8 y no ANSI. Si el archivo está en ANSI y declaras UTF-8 (o viceversa), los caracteres se rompen igual.

Meta viewport: tu página en el celular

Más de la mitad del tráfico web mundial viene de celulares. Sin el meta viewport, los navegadores móviles muestran tu página como si fuera de escritorio: la encogen para que quepa en la pantalla y el usuario tiene que hacer zoom con los dedos para leer. Es una experiencia terrible.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Esta etiqueta le dice al navegador móvil: "usa el ancho real del dispositivo y empieza con zoom al 100 por ciento". Con ella, tu página se adapta a la pantalla del telefono y el texto se lee sin zoom. width=device-width significa "el ancho de la página es igual al ancho del dispositivo", e initial-scale=1.0 significa "sin zoom inicial".

Es importante entender que el viewport no hace tu página responsive por si solo: el diseno adaptable lo logras con CSS. Pero sin el meta viewport, ni el mejor CSS responsive funciona en móviles, porque el navegador parte de una suposición incorrecta del tamaño de pantalla. Es el cimiento sobre el que se construye todo lo móvil.

Error común: copiar viewports raros de internet con maximum-scale=1.0 o user-scalable=no, que impiden al usuario hacer zoom. Eso es una mala práctica de accesibilidad: las personas con baja visión necesitan el zoom para leer. Usa el viewport estandar y no bloques el zoom del usuario.

Meta description y title: tu escaparate en Google

Cuando buscas algo en Google, cada resultado muestra tres cosas: un título azul clicable, una URL y un texto descriptivo debajo. El título azul viene de tu etiqueta title y el texto descriptivo viene (muchas veces) de tu meta description. Son tu escaparate: lo primero que un desconocido ve de tu sitio.

```
<title>Curso Completo de HTML5 - Lucas EGA Academy</title>  
<meta name="description" content="Aprende HTML5 desde cero con 30 clases gratuitas en español: estructura, formularios, validacion, semantica y SEO.">
```

El title es uno de los factores de posicionamiento más importantes. Debe ser descriptivo, incluir tu palabra clave principal y tu marca, y medir entre 50 y 60 caracteres. Si es más largo, Google lo corta con puntos suspensivos. Cada página de tu sitio debe tener un title único: no pongas "Inicio" en todas.

La meta description no es un factor directo de posicionamiento (Google lo ha confirmado), pero si afecta el porcentaje de clics: una descripción atractiva hace que más gente haga clic en tu resultado. Debe medir entre 150 y 160 caracteres, resumir el contenido de la página e incluir una llamada a la acción implícita. Piensalo como el anuncio gratis de tu página.

Un title generico como "Mi página" o "Bienvenidos" es una oportunidad desperdiciada. Compara: "Mi página" no le dice nada a nadie, mientras que "Curso Completo de HTML5 Gratis - Lucas EGA Academy" le dice a Google y al usuario exactamente que van a encontrar. Escribe titles para humanos primero y para Google después.

Keywords: la historia de una etiqueta muerta

En los años noventa y dos mil, la etiqueta meta keywords era importante: le decias a Google de que trataba tu página con una lista de palabras clave.

```
<meta name="keywords" content="html5, curso html, aprender html, tutorial html español">
```

El problema fue el abuso: los spammers llenaban keywords con cientos de palabras populares que no tenían nada que ver con su contenido para aparecer en búsquedas ajenas. Google se cansó y desde 2009 ignora oficialmente la meta keywords para posicionamiento. Hoy es una etiqueta muerta: no ayuda en nada y solo ocupa espacio.

La lección es valiosa: el SEO moderno no se trata de trucos ni de etiquetas mágicas, se trata de contenido real, estructura semántica clara y buena experiencia de usuario. Las keywords como concepto siguen vivas (investigar que palabras busca tu audiencia es fundamental), pero se trabajan en el contenido visible: en tus titles, encabezados y textos, no en una meta etiqueta invisible.

Si ves tutoriales viejos que recomiendan meta keywords, ya sabes que están desactualizados. No la uses, no pierdas tiempo en ella.

Open Graph: cómo se ve tu página en redes sociales

Cuando compartes un enlace en Facebook, WhatsApp, X o LinkedIn, aparece una tarjeta con imagen, título y descripción. Esa tarjeta no se genera por arte de magia: la controlas tu con las etiquetas Open Graph en el head.

```
<meta property="og:title" content="Curso Completo de HTML5 - Lucas EGA Academy">
<meta property="og:description" content="30 clases gratuitas para dominar HTML5 desde cero.">
<meta property="og:image" content="https://lucasegaacademy.com/img/curso-html5.png" property="og:image">
<meta property="og:url" content="https://lucasegaacademy.com/curso-html5">
<meta property="og:type" content="website">
```

Fijate que usan property en lugar de name: es la sintaxis del protocolo Open Graph creado por Facebook. Las cinco básicas son og:title, og:description, og:image, og:url y og:type. La imagen debe ser una URL absoluta (completa, con https) y de buen tamaño, mínimo 1200 por 630 pixeles para que se vea nítida.

Para X (Twitter) existen las Twitter Cards, que son parecidas:

```
<meta name="twitter:card" content="summary_large_image">
<meta name="twitter:title" content="Curso Completo de HTML5">
<meta name="twitter:description" content="30 clases gratuitas para dominar HTML5.">
<meta name="twitter:image" content="https://lucasegaacademy.com/img/curso-html5.png" name="twitter:image">
```

Si no pones Open Graph, las redes sociales adivinan que mostrar: a veces toman la primera imagen que encuentran (quizas tu logo pixeleado) y un fragmento de texto al azar. Con cinco líneas controlas tu imagen pública en redes. Para un creador de contenido como tu, que va a promocionar sus cursos en redes, esto es oro.

Jerarquía de encabezados y SEO

Los encabezados h1 a h6 no son solo texto grande: son la tabla de contenido que Google lee para entender tu página. La jerarquía correcta es:

- Un solo h1 por página: el tema principal. Es lo más importante.
- Varios h2: las secciones principales del tema.
- h3 dentro de cada h2: subsecciones.

- h4, h5, h6 para niveles más profundos (raramente necesarios).

```
<h1>Curso Completo de HTML5</h1>
<h2>Unidad 1: Estructura basica</h2>
<h3>La etiqueta html</h3>
<h3>La etiqueta head</h3>
<h2>Unidad 2: Texto y enlaces</h2>
<h3>Encabezados</h3>
```

Errores comunes que danan tu SEO: usar varios h1 en la misma página (confunde a Google sobre cual es el tema principal), saltarte niveles (un h4 directamente debajo de un h2), usar encabezados solo por su tamaño visual ("quiero texto grande" no es razón para un h2; para eso está CSS), y poner todo el texto en negritas con **b** en lugar de estructurarlo con encabezados.

La analogía del escaparate cierra el círculo: tu title y tu meta description son el letrero luminoso y el aparador de tu tienda en la avenida de Google. La jerarquía de encabezados es como están organizados los pasillos adentro. El contenido semántico es la mercancía bien etiquetada. Todo junto hace que los clientes (usuarios) entren, encuentren lo que buscan y se queden.

Errores comunes de la unidad y cómo evitarlos

Vamos a repasar los errores más frecuentes que cometen los estudiantes con estos tres temas, para que no caigas en ellos.

En validación: confiar solo en required y olvidar los formatos (un campo de correo con required pero sin type="email" acepta "asdf" como correo). Usar maxlength como medida de seguridad (no lo es). Poner pattern sin probarlo con datos reales. Marcar todo como obligatorio y espantar usuarios. Usar novalidate "porque me estorbaba el mensaje de error" en lugar de aprender a trabajar con la validación.

En semántica: el divitis que ya vimos. Usar section sin encabezado (una section debería llevar su h2 o h3). Confundir article con section: recuerda, article es independiente, section es temática. Poner el nav alrededor de cualquier lista de enlaces, incluyendo los del footer. Olvidar el main o poner dos mains. Usar header y footer solo una vez por "miedo", cuando en realidad puedes tener varios.

En SEO: títulos duplicados en todas las páginas. Meta descriptions de más de 160 caracteres que Google corta. Olvidar el viewport y quejarse de que "en mi celular se ve mal". No poner alt

en las imágenes (el alt es el SEO de las imágenes y la accesibilidad al mismo tiempo). Usar meta keywords pensando que ayuda. No revisar como se ve el enlace en WhatsApp por no poner Open Graph.

Buenas prácticas profesionales

Cierra la explicación con las buenas practicas que distinguen el código profesional.

Válida en capas: HTML nativo para la experiencia inmediata del usuario, JavaScript para reglas complejas e interactividad fina, y servidor siempre como última línea de defensa. Nunca confies solo en el navegador.

Escribe HTML semantico por defecto: header, nav, main, section, article, aside, footer. Usa div solo para envoltorios de presentación. Tu código debe leerse como un índice, no como una sopa de cajas.

Cuida tu head como cuidas tu fachada: charset UTF-8 primero, viewport siempre, title único y descriptivo por página, meta description atractiva de 150 a 160 caracteres, Open Graph completo con imagen de buena calidad. Revisa cada página nueva con esta lista de verificación.

Respetar la jerarquía de encabezados: un h1, luego h2, luego h3, sin saltos. Piensa en tus encabezados como el esquema que Google va a leer en voz alta para decidir de que trata tu página.

Prueba todo con datos reales: formularios con correos mal escritos, telefonos incompletos, contraseñas cortas; páginas vistas en celular y en computadora; enlaces compartidos en redes para ver la tarjeta Open Graph. La teoría sin prueba es solo una opinion.

Y documenta tus decisiones: un comentario HTML explicando por que un pattern es asi, o por que un campo lleva novalidate, le ahorra horas a tu yo del futuro y a cualquier companero que toque tu código.

Ejemplos paso a paso

Ejemplo 1: Formulario de registro con validación completa

Este es el ejemplo más importante de la unidad: un formulario de registro que usa todos los atributos de validación nativa que aprendiste. Analízalo línea por línea, porque este patron lo vas a reutilizar en casi todos tus proyectos.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Registro - Mi Sitio</title>
</head>
<body>
  <main>
    <h1>Crea tu cuenta gratis</h1>
    <!-- novalidate NO se usa aqui: queremos la validacion del navegador -->
    <form action="/registro" method="post">
      <!-- Nombre: obligatorio, solo letras con acentos, entre 2 y 50 caracteres -->
      <label for="nombre">Nombre completo:</label>
      <input type="text" id="nombre" name="nombre"
        pattern="[A-Za-zÁÉÍÓÚŮÑñáéíóú]{2,50}"
        title="Solo letras, entre 2 y 50 caracteres"
        required>
      <br><br>

      <!-- Correo: obligatorio, el navegador valida el formato -->
      <label for="correo">Correo electronico:</label>
      <input type="email" id="correo" name="correo"
        placeholder="tucorreo@ejemplo.com"
        required>
      <br><br>

      <!-- Contraseña: obligatoria, minimo 8 caracteres, letra y numero -->
      <label for="password">Contraseña:</label>
      <input type="password" id="password" name="password"
        pattern="(?=.*[A-Za-z])(?=.*[0-9]).{8,}"
        title="Minimo 8 caracteres, con al menos una letra y un numero"
        minlength="8"
        required>
      <br><br>

      <!-- Edad: obligatoria, entre 13 y 120 anos -->
      <label for="edad">Edad:</label>
      <input type="number" id="edad" name="edad"
        min="13" max="120"
        required>
      <br><br>

      <!-- Telefono: obligatorio, exactamente 10 digitos -->

```



```

<label for="telefono">Telefono (10 digitos):</label>
<input type="tel" id="telefono" name="telefono"
        pattern="[0-9]{10}"
        title="Escribe 10 digitos sin espacios ni guiones"
        required>
<br><br>

<!-- Codigo postal: opcional, pero si lo escriben deben ser 5 digitos -->
<label for="cp">Codigo postal (opcional):</label>
<input type="text" id="cp" name="cp"
        pattern="[0-9]{5}"
        title="5 digitos">
<br><br>

<!-- Terminos: obligatorio, el usuario debe aceptarlos -->
<input type="checkbox" id="terminos" name="terminos" required>
<label for="terminos">Acepto los terminos y condiciones</label>
<br><br>

<button type="submit">Crear cuenta</button>
<!-- Boton de borrador: envia sin validar -->
<button type="submit" formnovalidate>Guardar borrador</button>
</form>
</main>
</body>
</html>

```

Vamos a desglosar las decisiones de diseño. El campo nombre combina `pattern` con `{2,50}` para exigir entre 2 y 50 caracteres de letras, espacios y caracteres del español. Sin `required`, el `pattern` no se aplicaría a un campo vacío: recuerda que los campos opcionales vacíos siempre pasan la validación. Por eso el código postal es opcional (sin `required`): si el usuario lo deja vacío, pasa; si escribe algo, debe cumplir el `pattern` de 5 dígitos.

La contraseña usa dos capas: `minlength="8"` como mensaje claro y `pattern` para exigir letra y número. Podrías usar solo el `pattern`, pero `minlength` da un mensaje de error más amigable ("Utiliza al menos 8 caracteres") que el genérico del `pattern`.

El checkbox de términos con `required` es un truco clásico: obliga al usuario a marcar la casilla, porque un checkbox sin marcar se considera vacío. Sin `required`, el usuario podría registrarse sin aceptar los términos.

El boton "Guardar borrador" con formnovalidate permite enviar datos incompletos. Es el patron profesional para formularios largos: validación estricta para el envio final, flexibilidad para el borrador.

Prueba este formulario tu mismo: intenta enviarlo vacío, luego con un correo mal escrito, luego con un telefono de 9 digitos. Observa los mensajes del navegador, el borde rojo en los campos invalidos y como el foco salta al primer error. Esa es la validación nativa trabajando para ti.

Ejemplo 2: Estructura semántica completa de un blog

Ahora vamos a construir la estructura semantica de un blog de tecnología. Este ejemplo usa todas las etiquetas semanticas en su lugar correcto y te sirve como plantilla para casi cualquier sitio de contenido.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Blog de Tecnología - Lucas EGA Academy</title>
  <meta name="description" content="Articulos sobre programacion, HTML, CSS y desarrollo web en espanol.">
</head>
<body>
  <!-- CABECERA DEL SITIO: identidad + navegacion principal -->
  <header>
    <h1>Blog de Tecnologia</h1>
    <p>Aprende programacion en espanol</p>
    <nav>
      <ul>
        <li><a href="/">Inicio</a></li>
        <li><a href="/html">HTML</a></li>
        <li><a href="/css">CSS</a></li>
        <li><a href="/contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>

  <!-- CONTENIDO PRINCIPAL: unico por pagina -->
  <main>
    <!-- SECCION tematica que agrupa articulos -->
    <section>
      <h2>Ultimos articulos</h2>
```

```

<!-- ARTICULO independiente: tendria sentido por si solo -->
<article>
  <header>
    <h3>Validacion de formularios sin JavaScript</h3>
    <p>Publicado el <time datetime="2026-09-10">10 de septiembre de 2026</time>
por Lucas</p>
  </header>
  <p>HTML5 incluye atributos de validacion nativa que te ahorran cientos de
lineas de codigo...</p>
  <figure>
    
    <figcaption>El navegador muestra los errores automaticamente.</figcaption>
  </figure>
  <a href="/articulos/validacion">Leer articulo completo</a>
  <footer>
    <p>Etiquetas: HTML5, formularios, validacion</p>
  </footer>
</article>

<article>
  <header>
    <h3>HTML semantico: la guia definitiva</h3>
    <p>Publicado el <time datetime="2026-09-05">5 de septiembre de 2026</time>
por Lucas</p>
  </header>
  <p>Las etiquetas semanticas le dicen a Google exactamente que significa cada
parte de tu pagina...</p>
  <a href="/articulos/semantica">Leer articulo completo</a>
</article>
</section>

<!-- CONTENIDO COMPLEMENTARIO: relacionado pero no esencial -->
<aside>
  <h2>Sobre el autor</h2>
  <p>Lucas es desarrollador web y creador de Lucas EGA Academy.</p>
  <h2>Categorias populares</h2>
  <ul>
    <li><a href="/html">HTML (30 articulos)</a></li>
    <li><a href="/css">CSS (25 articulos)</a></li>
    <li><a href="/js">JavaScript (40 articulos)</a></li>
  </ul>
</aside>

```

```

</main>

<!-- PIE DEL SITIO: cierre, legal, contacto -->
<footer>
  <p>2026 Lucas EGA Academy. Todos los derechos reservados.</p>
  <p><a href="/privacidad">Aviso de privacidad</a> | <a
href="/terminos">Terminos</a></p>
</footer>
</body>
</html>

```

Observa los detalles finos. El header del sitio contiene el h1 (el tema principal de la página) y el nav con la navegación principal. Cada article tiene su propio header con el título y la fecha, y su propio footer con las etiquetas: esto es válido porque header y footer pueden usarse por sección, no solo una vez por página.

La etiqueta time con el atributo datetime es un detalle semantico hermoso: el texto visible dice "10 de septiembre de 2026" para humanos, y el atributo datetime="2026-09-10" le da a las máquinas la fecha en formato estandar. Google usa esto para saber cuando se publicó el contenido.

figure y figcaption envuelven la imagen con su pie de foto: le dicen al navegador que la imagen y su descripción forman una unidad. El atributo alt describe la imagen para los lectores de pantalla y para Google.

El aside está dentro de main pero fuera de section: es contenido complementario de la página, relacionado con el blog pero no parte de los artículos. Si lo quitaras, los artículos seguirían entendiendose perfecto. Esa es la prueba del aside.

La jerarquía de encabezados es impecable: un h1 para la página, h2 para las secciones principales ("Ultimos artículos", "Sobre el autor", "Categorías populares"), h3 para cada artículo. Sin saltos, sin duplicados.

Ejemplo 3: Head completo optimizado para SEO

El head es invisible para el visitante pero es lo primero que leen Google y las redes sociales. Este ejemplo reúne todo lo de la unidad en un head profesional listo para producción.

```

<head>
  <!-- 1. CODIFICACION: siempre primero, para que los acentos se lean bien -->
  <meta charset="UTF-8">

  <!-- 2. VIEWPORT: imprescindible para que se vea bien en celular -->

```

```

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<!-- 3. TITLE: unico por pagina, palabra clave + marca, 50-60 caracteres -->
<title>Curso Completo de HTML5 Gratis - Lucas EGA Academy</title>

<!-- 4. DESCRIPTION: tu anuncio en Google, 150-160 caracteres -->
<meta name="description" content="Aprende HTML5 desde cero con 30 clases gratuitas
en español: estructura, formularios, validacion, semantica y SEO. Empieza hoy mismo.">

<!-- 5. AUTOR: opcional pero profesional -->
<meta name="author" content="Lucas EGA Academy">

<!-- 6. OPEN GRAPH: como se ve tu pagina en Facebook, WhatsApp, LinkedIn -->
<meta property="og:title" content="Curso Completo de HTML5 Gratis - Lucas EGA
Academy">
<meta property="og:description" content="30 clases gratuitas para dominar HTML5
desde cero, en español.">
<meta property="og:image"
content="https://lucasegaacademy.com/img/curso-html5-og.png">
<meta property="og:url" content="https://lucasegaacademy.com/curso-html5">
<meta property="og:type" content="website">
<meta property="og:locale" content="es_MX">

<!-- 7. TWITTER CARDS: como se ve en X -->
<meta name="twitter:card" content="summary_large_image">
<meta name="twitter:title" content="Curso Completo de HTML5 Gratis">
<meta name="twitter:description" content="30 clases gratuitas para dominar HTML5
desde cero.">
<meta name="twitter:image"
content="https://lucasegaacademy.com/img/curso-html5-og.png">

<!-- 8. FAVICON: el iconito de la pestana del navegador -->
<link rel="icon" href="/favicon.ico" type="image/x-icon">

<!-- 9. IDIOMA: se declara en la etiqueta html, no en el head -->
<!-- <html lang="es"> : le dice a Google y a los lectores de pantalla el idioma -->
</head>

```

El orden importa: charset va primero porque el navegador necesita saber la codificación antes de leer lo demás. El viewport va segundo porque afecta como se renderiza todo. El title y la description son tu prioridad SEO.

Nota el og:locale con valor es_MX: le dice a las redes sociales que tu contenido está en español de México. Es un detalle que pocos ponen y que afina tu segmentación.

Las URLs de og:image y og:url son absolutas (completas con https). Esto es obligatorio: las redes sociales no pueden resolver rutas relativas como "/img/foto.png" porque leen tu head desde sus servidores. Si pones una ruta relativa, la tarjeta saldra sin imagen.

El favicon parece un detalle menor, pero las páginas sin favicon se ven descuidadas en las pestañas del navegador y en los marcadores. Es una línea que suma profesionalismo.

Lo que deliberadamente NO incluye este head: meta keywords (etiqueta muerta), viewports que bloquean el zoom (mala accesibilidad), y titles genericos. Cada línea aquí tiene un proposito probado.

Para verificar tu trabajo, usa estas herramientas gratuitas: el validador de tarjetas de X, el depurador de enlaces de Facebook, y la herramienta de inspección de URLs de Google Search Console. Pegar tu URL ahi te muestra exactamente como te ven las máquinas.

Ejemplo 4: Landing page semántica completa

Vamos a cerrar los ejemplos con una landing page real: página de venta del curso de HTML5 que combina estructura semantica, jerarquía de encabezados correcta y un formulario con validación. Es el ejemplo que une los tres temas de la unidad en un solo documento.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Curso Completo de HTML5 Gratis - Lucas EGA Academy</title>
  <meta name="description" content="Domina HTML5 con 30 clases gratuitas en español: estructura, formularios, validacion, semantica y SEO. Inscríbete gratis hoy.">
  <meta property="og:title" content="Curso Completo de HTML5 Gratis - Lucas EGA Academy">
  <meta property="og:description" content="30 clases gratuitas para dominar HTML5 desde cero.">
  <meta property="og:image" content="https://lucasegaacademy.com/img/curso-html5-og.png">
  <meta property="og:url" content="https://lucasegaacademy.com/curso-html5">
  <meta property="og:type" content="website">
</head>
<body>
  <header>
    <p><strong>Lucas EGA Academy</strong> - Aprende tecnologia en español</p>
    <nav>
      <ul>
```

```

        <li><a href="#temario">Temario</a></li>
        <li><a href="#testimonios">Testimonios</a></li>
        <li><a href="#registro">Inscribete</a></li>
    </ul>
</nav>
</header>

<main>
    <!-- SECCION hero: el mensaje principal -->
    <section>
        <h1>Curso Completo de HTML5</h1>
        <p>De cero a desarrollador web con 30 clases gratuitas en español.</p>
        <a href="#registro">Empezar gratis ahora</a>
    </section>

    <!-- SECCION temario: contenido tematico con subsecciones -->
    <section id="temario">
        <h2>Lo que vas a aprender</h2>
        <article>
            <h3>Validacion de formularios</h3>
            <p>El navegador valida por ti con required, pattern y tipos especiales.</p>
        </article>
        <article>
            <h3>HTML semantico</h3>
            <p>Estructura profesional que Google ama y posiciona mejor.</p>
        </article>
        <article>
            <h3>SEO basico</h3>
            <p>Meta etiquetas y encabezados para aparecer en Google.</p>
        </article>
    </section>

    <!-- SECCION testimonios -->
    <section id="testimonios">
        <h2>Lo que dicen los estudiantes</h2>
        <article>
            <h3>Maria G.</h3>
            <p>"Por fin entendi la validacion sin JavaScript. Directo y claro."</p>
        </article>
        <article>
            <h3>Carlos R.</h3>
            <p>"Mi pagina subio en Google solo con cambiar a etiquetas semanticas."</p>
        </article>
    </section>

```

```

<!-- SECCION registro: el formulario con validacion nativa -->
<section id="registro">
  <h2>Inscribete gratis</h2>
  <form action="/inscripcion" method="post">
    <label for="nombre-r">Nombre:</label>
    <input type="text" id="nombre-r" name="nombre"
      pattern="[A-Za-zÁÉÍÓÚŮÑñáéíóúü ]{2,50}" required>
    <br><br>
    <label for="correo-r">Correo electronico:</label>
    <input type="email" id="correo-r" name="correo" required>
    <br><br>
    <label for="nivel">Tu nivel actual:</label>
    <select id="nivel" name="nivel" required>
      <option value="">Elige una opcion</option>
      <option value="cero">Desde cero</option>
      <option value="basico">Basico</option>
      <option value="intermedio">Intermedio</option>
    </select>
    <br><br>
    <button type="submit">Quiero inscribirme</button>
  </form>
</section>

<aside>
  <h2>Tambien te puede interesar</h2>
  <ul>
    <li><a href="/curso-css">Curso de CSS</a></li>
    <li><a href="/curso-js">Curso de JavaScript</a></li>
  </ul>
</aside>
</main>

<footer>
  <p>2026 Lucas EGA Academy - lucasegaacademy.com</p>
  <p><a href="/privacidad">Aviso de privacidad</a></p>
</footer>
</body>
</html>

```

Analiza por que esta landing page es profesional. El nav usa enlaces de ancla (#temario, #testimonios, #registro) que saltan a las secciones con esos id: es el patron estandar de las landing pages de una sola página.

Cada section tiene su h2 y cada article su h3: la jerarquía es perfecta y Google entiende la estructura del contenido de un vistazo. El h1 aparece una sola vez con la propuesta de valor principal.

El formulario combina lo mejor de la unidad: nombre con pattern para letras en español, correo con type="email", y un select con required donde la primera opción tiene value vacío. Ese detalle del select es importante: si la primera opción tuviera un valor, el navegador consideraría el campo "lleno" y el required no serviría. Con value="" vacío, el usuario está obligado a elegir una opción real.

El aside con cursos relacionados está dentro de main: es contenido complementario que enriquece la página sin distraer del objetivo principal (la inscripción). Y el footer cierra con la marca y el aviso legal.

Esta landing page, tal como esta, ya es mejor que la mayoría de las páginas que vas a encontrar en internet: válida sola, habla el idioma de Google, se adapta al celular y tiene tarjetas para redes sociales. Todo con HTML puro.

Práctica

Ejercicio 1: Formulario de contacto validado

Enunciado: Crea un formulario de contacto con los siguientes campos, todos con validación nativa: nombre (obligatorio, solo letras en español, 2 a 40 caracteres), correo electrónico (obligatorio, formato válido), asunto (obligatorio, máximo 60 caracteres), mensaje (obligatorio, mínimo 20 caracteres, máximo 500) y un checkbox para aceptar el aviso de privacidad (obligatorio). El formulario debe enviarse a "/contacto" por POST. Agrega un boton secundario "Guardar borrador" que envíe sin validar.

Solución explicada paso a paso:

Paso 1: Abre la estructura básica del formulario con action y method. No pongas novalidate porque quieres la validación activa por defecto.

Paso 2: El campo nombre combina required con pattern para letras del español incluyendo espacios, y el rango {2,40} controla la longitud dentro del mismo pattern. Agrega title para orientar al usuario.

Paso 3: El correo usa type="email" con required. El navegador válida el formato automáticamente.

Paso 4: El asunto usa maxlength="60" para impedir textos larguissimos, más required.

Paso 5: El mensaje es un textarea con required, minlength="20" y maxlength="500". Recuerda que minlength y maxlength también funcionan en textarea.

Paso 6: El checkbox de privacidad lleva required para obligar a marcarlo.

Paso 7: Dos botones: el principal type="submit" normal y el secundario con formnovalidate para el borrador.

Código de la solución:

```
<form action="/contacto" method="post">
  <label for="nombre">Nombre:</label>
  <input type="text" id="nombre" name="nombre"
    pattern="[A-Za-zÁÉÍÓÚÛÑñáéíóúü ]{2,40}"
    title="Solo letras, entre 2 y 40 caracteres"
    required>
  <br><br>

  <label for="correo">Correo electronico:</label>
  <input type="email" id="correo" name="correo" required>
  <br><br>

  <label for="asunto">Asunto:</label>
  <input type="text" id="asunto" name="asunto" maxlength="60" required>
  <br><br>

  <label for="mensaje">Mensaje:</label>
  <textarea id="mensaje" name="mensaje" rows="5"
    minlength="20" maxlength="500" required></textarea>
  <br><br>

  <input type="checkbox" id="privacidad" name="privacidad" required>
  <label for="privacidad">Acepto el aviso de privacidad</label>
  <br><br>

  <button type="submit">Enviar mensaje</button>
  <button type="submit" formnovalidate>Guardar borrador</button>
</form>
```

Verificación: intenta enviar el formulario vacío y observa como el navegador bloquea el envío en el primer campo. Escribe un nombre con números y verifica que el pattern lo rechace. Escribe un mensaje de 10 caracteres y comprueba el mensaje de minlength. Marca todo bien y envía: el formulario pasa.

Ejercicio 2: Convierte una página con divitis a HTML semántico

Enunciado: Toma el siguiente código con divitis y reescribelo usando etiquetas semanticas. No cambies el contenido visible, solo las etiquetas. Identifica al menos 6 cambios.

```
<div id="cabecera">
  <div class="titulo">Mi Portafolio</div>
  <div class="menu">
    <a href="#proyectos">Proyectos</a>
    <a href="#contacto">Contacto</a>
  </div>
</div>
<div id="contenido">
  <div class="seccion">
    <div class="subtitulo">Proyectos destacados</div>
    <div class="proyecto">
      <div class="nombre-proyecto">Tienda en linea</div>
      <div class="descripcion">E-commerce con carrito de compras.</div>
    </div>
  </div>
</div>
<div id="pie">Contacto: hola@ejemplo.com</div>
```

Solución explicada paso a paso:

Paso 1: El div id="cabecera" es la cabecera del sitio: se convierte en header. El div class="titulo" con el nombre del sitio es el tema principal: se convierte en h1.

Paso 2: El div class="menu" agrupa la navegación principal: se convierte en nav. Los enlaces pueden ir directos o en una lista; la lista ul es la forma más semantica.

Paso 3: El div id="contenido" es el contenido principal único: se convierte en main. Solo debe haber uno, y aquí hay uno.

Paso 4: El div class="seccion" agrupa contenido tematico: se convierte en section, y el div class="subtitulo" en su encabezado h2.

Paso 5: El div class="proyecto" describe un trabajo independiente que tendría sentido por si solo: se convierte en article, con h3 para el nombre.

Paso 6: El div id="pie" es el pie: se convierte en footer.

Código de la solución:

```
<header>
  <h1>Mi Portafolio</h1>
  <nav>
    <ul>
      <li><a href="#proyectos">Proyectos</a></li>
      <li><a href="#contacto">Contacto</a></li>
    </ul>
  </nav>
</header>
<main>
  <section id="proyectos">
    <h2>Proyectos destacados</h2>
    <article>
      <h3>Tienda en linea</h3>
      <p>E-commerce con carrito de compras.</p>
    </article>
  </section>
</main>
<footer>
  <p>Contacto: hola@ejemplo.com</p>
</footer>
```

Verificación: cuenta los cambios: header, h1, nav, ul/li, main, section, h2, article, h3, p y footer. Son más de 6. La jerarquía de encabezados quedo en h1, h2, h3 sin saltos. El contenido visible es identico, pero ahora Google y los lectores de pantalla entienden la estructura.

Ejercicio 3: Arma un head SEO completo desde cero

Enunciado: Escribe el head completo para la página "Recetas de cocina mexicana" del sitio "Sabor Casero" (saborcasero.com). Debe incluir: charset UTF-8, viewport, title único y optimizado, meta description de 150 a 160 caracteres, meta author, las 5 etiquetas Open Graph básicas con imagen <https://saborcasero.com/img/recetas-og.png> y URL <https://saborcasero.com/recetas>, Twitter Card grande, y favicon. No incluyas meta keywords.

Solución explicada paso a paso:

Paso 1: charset UTF-8 como primera etiqueta para que los acentos de "cocina mexicana" se lean bien.

Paso 2: viewport estandar para móviles.

Paso 3: title con palabra clave al inicio y marca al final: "Recetas de Cocina Mexicana Fáciles - Sabor Casero". Cuenta los caracteres: queda dentro del rango 50 a 60.

Paso 4: description entre 150 y 160 caracteres que resuma y invite al clic. Cuéntala antes de darla por buena.

Paso 5: author con el nombre del sitio.

Paso 6: las 5 Open Graph: og:title, og:description, og:image, og:url, og:type. URLs absolutas con https.

Paso 7: Twitter Card con summary_large_image y sus etiquetas espejo.

Paso 8: link del favicon. Nada de meta keywords.

Código de la solución:

```

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Recetas de Cocina Mexicana Faciles - Sabor Casero</title>
  <meta name="description" content="Descubre recetas de cocina mexicana faciles y rapidas: tacos, enchiladas, pozole y mas. Ingredientes sencillos y paso a paso con fotos.">
  <meta name="author" content="Sabor Casero">
  <meta property="og:title" content="Recetas de Cocina Mexicana Faciles - Sabor Casero">
  <meta property="og:description" content="Tacos, enchiladas, pozole y mas recetas mexicanas faciles con fotos paso a paso.">
  <meta property="og:image" content="https://saborcasero.com/img/recetas-og.png">
  <meta property="og:url" content="https://saborcasero.com/recetas">
  <meta property="og:type" content="website">
  <meta name="twitter:card" content="summary_large_image">
  <meta name="twitter:title" content="Recetas de Cocina Mexicana Faciles - Sabor Casero">
  <meta name="twitter:description" content="Tacos, enchiladas, pozole y mas recetas mexicanas faciles con fotos paso a paso.">
  <meta name="twitter:image" content="https://saborcasero.com/img/recetas-og.png">
  <link rel="icon" href="/favicon.ico" type="image/x-icon">
</head>

```

Verificación: revisa la lista mentalmente. Charset primero: si. Viewport: si. Title único con keyword y marca: si. Description en rango: cuentalo, debe dar entre 150 y 160. Open Graph con URLs absolutas: si. Sin keywords: correcto. Este head está listo para producción.

Ejercicio 4: Válida un formulario de reservación con fechas y números

Enunciado: Crea un formulario de reservación de hotel con: nombre del huesped (obligatorio, letras en español), correo (obligatorio), fecha de llegada (obligatoria, no puede ser anterior al 2026-01-01 ni posterior al 2026-12-31), número de huespedes (obligatorio, entre 1 y 6), número de noches (obligatorio, mínimo 1, máximo 30) y tipo de habitación (select obligatorio con opciones Sencilla, Doble y Suite). Usa los atributos correctos para cada tipo de dato.

Solución explicada paso a paso:

Paso 1: El nombre usa el pattern de letras en español que ya dominas, con required.

Paso 2: El correo usa type="email" con required.

Paso 3: La fecha usa `type="date"` con `min="2026-01-01"` y `max="2026-12-31"`. El formato de fecha en los atributos es AAAA-MM-DD, siempre. El navegador muestra un calendario que bloquea las fechas fuera del rango.

Paso 4: El número de huéspedes usa `type="number"` con `min="1"` y `max="6"`. Las flechitas del navegador respetan el rango.

Paso 5: El número de noches usa `type="number"` con `min="1"` y `max="30"`.

Paso 6: El select lleva `required` y su primera opción tiene `value=""` vacío para obligar a elegir.

Paso 7: No uses `minlength` ni `maxlength` aquí: son para longitud de texto, no para valores numéricos ni fechas. Ese es el punto clave del ejercicio.

Código de la solución:

```

<form action="/reservar" method="post">
  <label for="huesped">Nombre del huesped:</label>
  <input type="text" id="huesped" name="huesped"
    pattern="[A-Za-zÁÉÍÓÚÜÑñáéíóúü ]{2,50}"
    title="Solo letras, entre 2 y 50 caracteres"
    required>
  <br><br>

  <label for="correo">Correo electronico:</label>
  <input type="email" id="correo" name="correo" required>
  <br><br>

  <label for="llegada">Fecha de llegada (2026):</label>
  <input type="date" id="llegada" name="llegada"
    min="2026-01-01" max="2026-12-31" required>
  <br><br>

  <label for="personas">Numero de huespedes (1 a 6):</label>
  <input type="number" id="personas" name="personas"
    min="1" max="6" required>
  <br><br>

  <label for="noches">Numero de noches (1 a 30):</label>
  <input type="number" id="noches" name="noches"
    min="1" max="30" required>
  <br><br>

  <label for="habitacion">Tipo de habitacion:</label>
  <select id="habitacion" name="habitacion" required>
    <option value="">Elige una opcion</option>
    <option value="sencilla">Sencilla</option>
    <option value="doble">Doble</option>
    <option value="suite">Suite</option>
  </select>
  <br><br>

  <button type="submit">Reservar</button>
</form>

```

Verificación: prueba elegir una fecha de 2025 en el calendario: el navegador no te deja. Escribe 8 huespedes: el navegador lo rechaza al enviar. Intenta enviar sin elegir habitación: el required del select lo bloquea porque la opción elegida tiene valor vacío. Cada atributo válida exactamente el tipo de dato que le corresponde.

Ejercicio 5: Detecta y corrige 8 errores en este código

Enunciado: El siguiente código tiene 8 errores relacionados con los temas de la unidad (validación, semántica y SEO). Encuéntralos todos, explica por qué son errores y escribe la versión corregida.

```
<head>
  <meta name="keywords" content="hotel, reservacion, vacaciones, playa, hotel,
hotel">
  <title>Mi pagina</title>
</head>
<body>
  <div id="header">
    <h1>Hotel Paraiso</h1>
    <h1>Reserva tus vacaciones</h1>
  </div>
  <div id="main">
    <input type="text" name="telefono" min="10" max="10" required>
  </div>
  <form action="/buscar" novalidate>
    <input type="email" name="correo" required>
    <button type="submit">Buscar</button>
  </form>
</body>
```

Solución explicada paso a paso:

Error 1: meta keywords. Es una etiqueta muerta que Google ignora desde 2009. Se elimina y se reemplaza por una meta description útil.

Error 2: Falta meta charset="UTF-8". Sin el, los acentos de "Paraiso" y "vacaciones" pueden romperse. Se agrega como primera etiqueta del head.

Error 3: Falta meta viewport. La página se vera mal en celular. Se agrega el viewport estandar.

Error 4: title generico "Mi página". No dice nada a Google ni al usuario. Se reemplaza por un title descriptivo con palabra clave y marca.

Error 5: Dos h1 en la misma página ("Hotel Paraiso" y "Reserva tus vacaciones"). Confunde a Google sobre el tema principal. El segundo debe ser h2 o un párrafo destacado.

Error 6: div id="header" y div id="main" en lugar de las etiquetas semanticas header y main. Se reemplazan por las etiquetas reales.

Error 7: input type="text" con min="10" max="10". Los atributos min y max no funcionan en campos de texto, solo en numericos y fechas. Para un telefono de 10 digitos lo correcto es type="tel" con pattern="[0-9]{10}".

Error 8: novalidate en el formulario de busqueda sin justificación. Desactiva la validación del correo sin motivo. Se elimina para que el navegador valide el email.

Código corregido:

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Hotel Paraiso - Reserva tus vacaciones en la playa</title>
  <meta name="description" content="Reserva tus vacaciones en Hotel Paraiso:
habitaciones frente a la playa, precios directos y confirmacion inmediata.">
</head>
<body>
  <header>
    <h1>Hotel Paraiso</h1>
    <p>Reserva tus vacaciones</p>
  </header>
  <main>
    <label for="telefono">Telefono (10 digitos):</label>
    <input type="tel" id="telefono" name="telefono"
      pattern="[0-9]{10}"
      title="Escribe 10 digitos sin espacios" required>
  </main>
  <form action="/buscar">
    <label for="correo">Correo para enviarte la cotizacion:</label>
    <input type="email" id="correo" name="correo" required>
    <button type="submit">Buscar</button>
  </form>
</body>
</html>
```

Verificación: repasa la lista de 8 errores y confirma que cada uno quedo corregido. Este ejercicio es un simulacro de revisión de código real: en el trabajo profesional, detectar estos errores en el código de otros (y en el propio) es una habilidad que vale oro.

Ejercicio 6: Página de artículo con todos los temas integrados

Enunciado: Crea la página completa de un artículo de blog que integre los tres temas de la unidad: estructura semántica (header con nav, main con article y aside, footer), head optimizado para SEO (charset, viewport, title, description, Open Graph básico), jerarquía de encabezados correcta, y al final del artículo un formulario de comentarios con validación (nombre obligatorio con letras en español, correo obligatorio con formato válido, comentario obligatorio de mínimo 10 caracteres).

Solución explicada paso a paso:

Paso 1: Escribe el doctype y la etiqueta html con lang="es".

Paso 2: Arma el head: charset primero, viewport, title único con la keyword del artículo, description de 150 a 160 caracteres, y las 5 Open Graph con URLs absolutas.

Paso 3: En el body, abre con header que contenga el nombre del blog y el nav principal.

Paso 4: El main contiene un article con su propio header (título h2 del artículo, fecha con time), párrafos de contenido con subtítulos h3, y un aside con artículos relacionados.

Paso 5: Después del article pero dentro de main, agrega una section con el formulario de comentarios validado.

Paso 6: Cierra con el footer del sitio.

Paso 7: Verifica la jerarquía: un solo h1 (nombre del blog en el header), h2 para el título del artículo y las secciones, h3 para subtítulos.

Código de la solución:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>10 trucos de HTML5 que todo principiante debe saber - Blog Tech</title>
  <meta name="description" content="Descubre 10 trucos de HTML5 para principiantes:
validacion nativa, etiquetas semanticas y SEO basico explicados con ejemplos practicos
en espanol.">
  <meta property="og:title" content="10 trucos de HTML5 que todo principiante debe
saber">
  <meta property="og:description" content="Validacion nativa, semantica y SEO
explicados con ejemplos practicos.">
  <meta property="og:image" content="https://blogtech.com/img/trucos-html5.png">
```

```

<meta property="og:url" content="https://blogtech.com/trucos-html5">
<meta property="og:type" content="article">
</head>
<body>
  <header>
    <h1>Blog Tech</h1>
    <nav>
      <ul>
        <li><a href="/">Inicio</a></li>
        <li><a href="/html">HTML</a></li>
        <li><a href="/css">CSS</a></li>
      </ul>
    </nav>
  </header>

  <main>
    <article>
      <header>
        <h2>10 trucos de HTML5 que todo principiante debe saber</h2>
        <p>Publicado el <time datetime="2026-09-19">19 de septiembre de
2026</time></p>
      </header>
      <section>
        <h3>Truco 1: validacion sin JavaScript</h3>
        <p>El atributo required convierte cualquier campo en obligatorio...</p>
      </section>
      <section>
        <h3>Truco 2: usa etiquetas semanticas</h3>
        <p>Reemplaza tus divs genericos por header, nav, main y footer...</p>
      </section>
      <footer>
        <p>Escrito por el equipo de Blog Tech</p>
      </footer>
    </article>

    <section>
      <h2>Deja tu comentario</h2>
      <form action="/comentar" method="post">
        <label for="nombre-c">Nombre:</label>
        <input type="text" id="nombre-c" name="nombre"
          pattern="[A-Za-zÁÉÍÓÚŨÑñáéíóúü ]{2,40}" required>
        <br><br>
        <label for="correo-c">Correo (no se publica):</label>
        <input type="email" id="correo-c" name="correo" required>

```

```

        <br><br>
        <label for="comentario">Comentario:</label>
        <textarea id="comentario" name="comentario"
            minlength="10" maxlength="500" required></textarea>
        <br><br>
        <button type="submit">Publicar comentario</button>
    </form>
</section>

<aside>
    <h2>Articulos relacionados</h2>
    <ul>
        <li><a href="/formularios">Guia de formularios HTML5</a></li>
        <li><a href="/seo">SEO basico para desarrolladores</a></li>
    </ul>
</aside>
</main>

<footer>
    <p>2026 Blog Tech. Todos los derechos reservados.</p>
</footer>
</body>
</html>

```

Verificación final de la unidad: un solo h1, jerarquía sin saltos, charset y viewport presentes, description en rango, Open Graph con URLs absolutas, article independiente con su header y footer propios, aside complementario, formulario con required, type="email", pattern y minlength. Si tu página cumple todo esto, dominas la unidad 5.

Resumen de la unidad

- La validación nativa de HTML5 permite que el navegador valide formularios sin JavaScript, con mensajes de error automaticos en el idioma del usuario.
- El atributo required convierte cualquier campo en obligatorio; el navegador bloquea el envio y resalta el campo si esta vacío.
- minlength y maxlength controlan la longitud del texto: maxlength limita mientras se escribe y minlength se válida al enviar.
- min, max y step controlan valores numericos y fechas; solo funcionan en inputs de tipo number, range, date y similares, nunca en texto.

- Los tipos email, url y number validan formatos automáticamente; tel muestra el teclado telefonico en celulares pero no válida por si solo.
- El atributo pattern válida con expresiones regulares: [A-Za-z]+ para letras, [0-9]{10} para 10 dígitos, y se combina con title para orientar al usuario.
- Las pseudo clases CSS:valid e:invalid permiten pintar los campos de verde o rojo en tiempo real segun el estado de la validación.
- novalidate en el form desactiva toda la validación del navegador; formnovalidate lo hace solo para un boton específico, ideal para "guardar borrador".
- La validación del navegador es la primera línea de defensa, no la única: en proyectos reales siempre se válida también en el servidor.
- HTML semantico significa usar etiquetas que describen el significado del contenido: header, nav, main, section, article, aside y footer.
- Google usa las etiquetas semanticas para entender la estructura de tu página y posicionarla mejor; los lectores de pantalla las usan para navegar.
- Solo debe haber un main por página; header y footer pueden repetirse por sección; article es contenido independiente y section es agrupación tematica.
- El divitis (usar div para todo) produce código mudo que ni Google ni los lectores de pantalla entienden; reserva div para envoltorios de presentación.
- meta charset="UTF-8" va primero en el head y garantiza que acentos, ñes y caracteres especiales se vean bien en todo el mundo.
- meta viewport con width=device-width es obligatorio para que la página se adapte al celular; nunca bloquee el zoom del usuario.
- El title (50 a 60 caracteres, único por página) y la meta description (150 a 160 caracteres) son tu escaparate en los resultados de Google.
- La meta keywords está muerta desde 2009: Google la ignora; no la uses.
- Las etiquetas Open Graph (og:title, og:description, og:image, og:url, og:type) controlan como se ve tu página al compartirla en redes sociales, con URLs absolutas.
- La jerarquía de encabezados es un solo h1 por página, luego h2, luego h3, sin saltos ni duplicados; es la tabla de contenido que Google lee.

UNIDAD 6: Multimedia y tablas profesionales

Bienvenido a la unidad 6 del Curso Completo de HTML5 de Lucas EGA Academy. En esta unidad vas a aprender a darle vida a tus páginas web con contenido multimedia real: audio y video que se reproducen directo en el navegador, tablas profesionales con encabezados y celdas unidas, y contenido embebido de otros sitios como videos de YouTube o mapas de Google. Todo esto sin plugins, sin programas extra y sin complicaciones: es puro HTML5.

Hasta ahora, en las unidades anteriores, construiste la estructura de tus páginas: textos, encabezados, listas, enlaces e imágenes. Tus páginas ya se ven bien y se entienden bien. Pero seamos honestos: una página que solo tiene texto e imágenes se siente estática, como un periodico impreso. El internet moderno es multimedia: la gente escucha música, ve videos, consulta tablas de datos y explora mapas interactivos. Si quieres que tus páginas se sientan como las páginas profesionales que visitas todos los días, necesitas dominar exactamente lo que cubre esta unidad.

Esta unidad cubre las clases 16, 17 y 18 del curso, y cada una abre una puerta importante. La clase 16 te ensena la multimedia nativa de HTML5: con `<audio controls src="musica.mp3">` y `<video controls src="video.mp4">` pones música y video directo en tu página, sin plugins. Antes de HTML5, reproducir video en una web era un dolor de cabeza: necesitabas Flash o reproductores externos que muchas veces fallaban. HTML5 cambio eso para siempre con etiquetas nativas que el navegador entiende sin ayuda de nadie. La clase 17 te lleva a las tablas profesionales: `thead` para el encabezado, `tbody` para el cuerpo, `tfoot` para el pie, y `th` con `colspan="2"` y `rowspan="2"` para unir celdas. Con esto tus tablas dejan de ser cuadradas básicas y se convierten en tablas de datos de verdad, como las de horarios, precios o reportes. Y la clase 18 te muestra el poder de `<iframe src="...">`: con esta etiqueta incrustas otra página dentro de la tuya, así puedes poner un mapa de Google o un video de YouTube directo en tu web, como si fuera parte de ella.

Por que importa esta unidad? Porque es la diferencia entre una página que se lee y una página que se vive. Piensa en las páginas que mas visitas: YouTube es video, Spotify es audio, Wikipedia esta llena de tablas de datos, y casi cualquier negocio serio tiene un mapa embebido con su ubicación. Todo eso se hace con las etiquetas de esta unidad. Además, estas habilidades son de las mas pedidas cuando alguien te pide una página: "quiero que mi página tenga un video de presentación", "necesito una tabla con mis precios", "quiero que aparezca mi negocio en un mapa". Después de esta unidad vas a poder decir que si a todo eso.

Como encaja en el curso? Las unidades anteriores te dieron los cimientos: estructura, texto, enlaces e imágenes. Esta unidad agrega la capa de riqueza: multimedia y datos tabulados. Las unidades siguientes van a construir sobre esto: cuando aprendas formularios, por ejemplo, podras combinar un formulario de contacto con un mapa embebido de tu negocio, que es exactamente lo que hacen las páginas de contacto profesionales. Y cuando llegues a CSS, vas a poder darle estilo a tus reproductores y tablas para que se vean espectaculares. Nada de lo que aprendas aquí se queda aislado: todo se conecta.

Una advertencia honesta antes de empezar: esta unidad tiene muchos atributos y detalles (controls, autoplay, muted, preload, colspan, rowspan, sandbox...). No intentes memorizarlos todos de golpe. Lo importante es que entiendas la idea de cada etiqueta y sepas donde buscar el atributo cuando lo necesites. La practica de los ejercicios al final va a fijar lo esencial. Vamos paso a paso, con calma, y al final de la unidad vas a mirar tus páginas con otros ojos.

Explicación detallada

Audio y video nativos: adiós a los plugins

Vamos a empezar con un poco de historia para que valores lo que HTML5 te regala. Hace años, si querías poner un video en tu página web, necesitabas un plugin llamado Flash. El visitante tenía que tenerlo instalado, actualizado y activado. Si no lo tenía, veía un cuadro vacío o un mensaje de error. En celulares era peor: muchos ni siquiera soportaban Flash. Era frágil, inseguro y molesto. HTML5 dijo "basta" y creó dos etiquetas que los navegadores entienden de forma nativa, sin plugins: `<audio>` y `<video>`. Nativa significa que viene incluida en el navegador, como las llantas vienen incluidas en un carro nuevo: no tienes que comprar nada extra.

La etiqueta `<audio>` sirve para reproducir sonido: música, podcasts, efectos, narraciones. Su forma más simple es esta: le pones el atributo controls para que aparezca el reproductor con los botones de play, pausa y volumen, y el atributo src con la ruta del archivo de sonido. Así de directo:

Con `<audio controls src="musica.mp3">` el navegador muestra un reproductor y el visitante puede darle play a tu archivo musica.mp3. Sin JavaScript, sin plugins, sin magia. Lo mismo pasa con `<video>`: con `<video controls src="video.mp4">` muestras un reproductor de video con la imagen del video y los controles. Fíjate qué elegantes son: la etiqueta dice lo que hace. Eso es HTML semántico en acción.

Ahora, hablemos de los formatos, porque aquí hay un detalle importante del mundo real. No todos los navegadores reproducen todos los formatos de audio y video. Es como los idiomas: no toda la gente habla todos los idiomas, así que si quieres que te entiendan en varios países, a veces necesitas hablar más de uno. En audio, los formatos más comunes son mp3, ogg y wav. El mp3 es el más universal: casi todo lo reproduce. El ogg es libre y de buena calidad, y el wav es audio sin compresión (pesa mucho, se usa poco en web). En video, los formatos comunes son mp4, webm y ogv. El mp4 es el rey: lo reproduce prácticamente todo. El webm es libre y eficiente, impulsado por Google.

¿Cómo te aseguras de que tu audio o video se escuche en todos los navegadores? Con la etiqueta `<source>`. En lugar de poner un solo archivo en el atributo `src`, pones varias etiquetas `<source>` dentro de `<audio>` o `<video>`, cada una con un formato distinto. El navegador prueba la primera, y si no la entiende, pasa a la siguiente. Es tu plan B automático. Piensa en `<source>` como llevar dos llaves de tu casa: si una no abre, pruebas la otra, sin dramas. Además, cada `<source>` lleva el atributo `type` que dice qué tipo de archivo es, por ejemplo `type="audio/mpeg"` para mp3 o `type="video/mp4"` para mp4. Esto ayuda al navegador a decidir rápido sin tener que descargar el archivo para averiguarlo.

Otro detalle profesional: siempre pon un texto alternativo dentro de `<audio>` o `<video>`, después de las etiquetas `<source>`. Ese texto solo se muestra si el navegador no puede reproducir ninguno de los formatos. Es tu red de seguridad final, como el mensaje que dejas en la puerta cuando no estás: "Tu navegador no soporta audio HTML5". Poca gente lo ve, pero cuando se necesita, se agradece.

Vamos con los atributos que controlan el comportamiento del reproductor, porque aquí es donde pasas de "funciona" a "se siente profesional". Ya conoces controls: sin él, el audio o video existe pero el visitante no tiene botones para manejarlo (a veces se usa así cuando controlas la reproducción con JavaScript, pero eso es tema avanzado). El atributo `width` y `height` en `<video>` definen el tamaño del reproductor en píxeles. Ojo: es mejor controlar el tamaño final con CSS más adelante, pero `width` y `height` ayudan a que la página reserve el espacio desde el inicio y no brinque el contenido cuando el video carga.

El atributo `autoplay` hace que el audio o video empiece solo al cargar la página. Suena atractivo, pero aquí viene una de las lecciones más importantes de esta unidad: los navegadores modernos BLOQUEAN el `autoplay` con sonido. ¿Por qué? Porque a nadie le gusta abrir una página y que de pronto suene música a todo volumen sin haberle dado play. Es una de las experiencias más molestas del internet, y los navegadores la prohibieron. Entonces, ¿`autoplay` ya no sirve? Sí sirve, pero con una condición: combinado con `muted`. Un video con `autoplay muted` (empieza solo pero en silencio) sí está permitido, y es el truco que

usan las portadas de muchos sitios: un video de fondo que se mueve solo, sin sonido, elegante y sin molestar. Si el visitante quiere sonido, le da play él mismo. Regla de oro: nunca fuerces sonido sin permiso del visitante.

El atributo `loop` hace que el audio o video se repita automáticamente cuando termina. Útil para música de ambiente o videos de fondo cortos. El atributo `muted` inicia el contenido en silencio (el visitante puede activar el sonido con los controles). El atributo `preload` le dice al navegador qué tanto debe descargar antes de que el visitante le dé play: puede ser `none` (no descarga nada hasta que le den play, ahorra datos), `metadata` (descarga solo datos básicos como la duración) o `auto` (descarga lo que pueda). Para una página con muchos videos, `preload="none"` o `metadata` cuidan la velocidad de carga y el plan de datos de tus visitantes de celular.

El atributo `poster` es exclusivo de `<video>` y es un detalle que separa lo amateur de lo profesional: es la imagen que se muestra antes de que el visitante le dé play. Sin `poster`, el navegador muestra el primer cuadro del video, que muchas veces es un cuadro negro o una imagen a la mitad de un movimiento, poco atractiva. Con `poster="portada.jpg"` muestras una imagen bonita, con título o invitación a darle play, como la portada de una película en Netflix. Siempre que pongas un video importante, ponle `poster`.

Y llegamos a una joya de accesibilidad: los subtítulos con la etiqueta `<track>`. Dentro de `<video>` puedes agregar `<track>` para subtítulos, y esto es importantísimo por dos razones: hay personas con discapacidad auditiva que dependen de los subtítulos, y hay muchísima gente que ve videos en silencio (en el transporte, en el trabajo). Los subtítulos se escriben en un archivo de texto con formato WebVTT (extensión.vtt), donde cada subtítulo lleva el tiempo en que aparece y desaparece. La etiqueta se ve así: `<track src="subtitulos.vtt" kind="subtitles" srclang="es" label="Español">`. El atributo `kind` dice qué tipo de pista es (subtitles para subtítulos), `srclang` el idioma (es para español) y `label` el nombre que verá el visitante en el menú del reproductor. Puedes poner varias pistas en varios idiomas y el visitante elige. Esto no es un lujo: en muchos contextos es lo que hace tu video usable para todos.

Errores comunes con audio y video que debes evitar: primero, poner `autoplay` con sonido y esperar que funcione (no va a funcionar, el navegador lo bloquea). Segundo, usar un solo formato raro y que en algunos navegadores no se reproduzca nada (usa `mp3/mp4` como base y agrega `<source>` de respaldo). Tercero, olvidar el atributo `controls` y que el visitante no pueda darle play (a menos que lo controles con JavaScript a propósito). Cuarto, no poner texto alternativo dentro de la etiqueta. Y quinto, subir archivos gigantescos sin comprimir: un video de 500 MB tarda una eternidad en cargar; comprime tus archivos antes de subirlos.

Buenas prácticas: usa autoplay solo con muted y solo cuando aporte algo (un video de fondo, no un comercial a gritos). Pon poster en tus videos importantes. Ofrece subtítulos cuando el video tenga diálogo importante. Usa preload="metadata" o "none" si la página tiene varios archivos multimedia. Y comprime: un mp3 a 128 kbps suena bien para web, y un video en mp4 con buena compresión puede pesar diez veces menos sin que se note la diferencia.

Tablas avanzadas: de cuadrícula básica a tabla profesional

Ya conoces lo básico de tablas de unidades anteriores: `<table>` para la tabla, `<tr>` para cada fila, `<td>` para cada celda y `<th>` para celdas de encabezado. Eso te sirve para tablas sencillas. Pero las tablas del mundo real (horarios de clases, listas de precios, reportes de ventas, tablas de posiciones) necesitan más estructura. HTML te da etiquetas para organizar una tabla como un profesional: `thead`, `tbody` y `tfoot`.

Piensa en una tabla como en un libro: tiene portada, capítulos y contraportada. El `thead` (table head) es el encabezado de la tabla: ahí van los títulos de las columnas, usando `<th>`. El `tbody` (table body) es el cuerpo: ahí van los datos, fila por fila, usando `<td>`. Y el `tfoot` (table foot) es el pie: ahí van los totales, resúmenes o notas finales. Separar la tabla en estas tres partes no cambia cómo se ve a simple vista, pero cambia todo por dentro: los lectores de pantalla (programas que leen páginas en voz alta para personas ciegas) entienden mejor la tabla, los navegadores pueden repetir el encabezado si la tabla se imprime en varias páginas, y cuando aprendas CSS podrás darle estilo distinto a cada parte con una sola regla.

Un detalle curioso que te puede sorprender: en el código HTML, el `tfoot` se escribe ANTES del `tbody`, aunque se muestra al final. Es una regla histórica del estándar: el pie se declara antes del cuerpo para que el navegador pueda mostrar los totales sin esperar a que carguen todos los datos. No te preocupes si se siente raro al principio; con la práctica se vuelve natural. El orden en tu código es: `thead`, `tfoot`, `tbody`. El orden en pantalla es: encabezado, cuerpo, pie.

Dentro del thead, las celdas deben ser `<th>`, no `<td>`. La etiqueta `<th>` significa "celda de encabezado" (table header) y por defecto se muestra en negritas y centrada, pero lo importante no es cómo se ve sino lo que significa: le dice al navegador y a los lectores de pantalla "esto es un título, no un dato". Y aquí entra el atributo `scope`, un pequeño gran detalle de accesibilidad: `scope="col"` en un `<th>` dice "este encabezado describe toda la columna que está debajo", y `scope="row"` dice "este encabezado describe toda la fila que está a su derecha". Imagina una tabla de horarios donde la primera columna tiene los días (Lunes, Martes...): esos `<th>` llevan `scope="row"` porque encabezan filas. Los títulos superiores (Hora, Materia, Salón) llevan `scope="col"`. Con `scope`, una persona que escucha la tabla con un lector de pantalla entiende qué dato pertenece a qué encabezado. Es un atributo pequeño que hace tu tabla usable para todos.

La etiqueta `<caption>` es el título oficial de la tabla. Se escribe justo después de abrir `<table>` y describe de qué trata la tabla, por ejemplo "Horario de clases del grupo A". A diferencia de poner un `<h3>` encima de la tabla, el `<caption>` está ligado a la tabla: los lectores de pantalla lo anuncian junto con ella. Es como el título de un libro impreso en la portada, no como una nota adhesiva pegada encima.

Ahora la parte divertida: unir celdas con `colspan` y `rowspan`. A veces una celda necesita ocupar el espacio de dos o más. Por ejemplo, en una tabla de horarios, si el recreo dura dos periodos seguidos, no quieres dos celdas que digan "Recreo" y "Recreo": quieres UNA celda ancha que diga "Recreo" abarcando ambas columnas. Eso se hace con `colspan="2"` (column span: la celda se extiende 2 columnas). Y si una materia dura dos horas seguidas en el mismo salón, quieres una celda alta que abarque dos filas: eso es `rowspan="2"` (row span: la celda se extiende 2 filas).

La analogía perfecta es Excel o Google Sheets: cuando seleccionas varias celdas y les das "combinar celdas", eso es exactamente `colspan` y `rowspan`. La diferencia es que en HTML lo haces con un atributo. Pero ojo con la trampa más común: cuando unes celdas, las filas afectadas deben tener MENOS celdas `<td>`, porque la celda unida ya ocupa ese espacio. Si una fila normal tiene 4 celdas y en la siguiente unes dos con `colspan="2"`, esa fila solo lleva 3 etiquetas `<td>` (la unida cuenta por dos). Si te equivocas en la cuenta, la tabla se descuadra y se ve rota. Mi consejo: dibuja tu tabla en papel primero, marca qué celdas se unen, y luego escribe el código contando con cuidado.

La etiqueta `<colgroup>` (con `<col>` adentro) es un extra elegante: te permite aplicar algo a columnas completas sin repetirlo en cada celda. Por ejemplo, si quieres que la primera columna tenga cierto ancho o estilo, lo defines una vez en `<colgroup>`. Es como decir "esta regla aplica a toda la columna 1" en lugar de ir celda por celda. No es obligatoria, pero cuando tus tablas crezcan la vas a agradecer.

Errores comunes con tablas: primero, usar solo `<td>` en todas partes y ningún `<th>` (la tabla "funciona" pero nadie, ni los lectores de pantalla ni los buscadores, sabe cuáles son los encabezados). Segundo, olvidar `thead/tbody/tfoot` en tablas de datos reales (pierdes estructura y accesibilidad). Tercero, equivocarse en la cuenta de celdas al usar `colspan` o `rowspan` (la tabla se descuadra). Cuarto, usar tablas para acomodar el diseño de la página (esto se hacía en los años 90; hoy el diseño se hace con CSS, las tablas son solo para datos tabulares de verdad). Y quinto, tablas gigantes sin `<caption>` ni `scope`, imposibles de entender para quien usa lector de pantalla.

Buenas prácticas: usa `<th>` con `scope` en todos tus encabezados. Divide siempre tus tablas de datos en `thead`, `tbody` y `tfoot`. Pon `<caption>` descriptivo. Dibuja en papel las celdas unidas antes de programarlas. Y recuerda: si lo que quieres mostrar NO son datos en filas y columnas (por ejemplo, acomodar un menú al lado del contenido), no uses tabla: eso se resuelve con CSS en unidades posteriores.

iframe y contenido embebido: tu página dentro de otra página

La etiqueta `<iframe>` es una de las más poderosas de HTML y también una de las más simples de usar. Su idea es esta: muestra OTRA página web dentro de la tuya, como una ventana. Con `<iframe src="...">` incrustas una página externa en tu sitio: puede ser un video de YouTube, un mapa de Google Maps, un documento, o casi cualquier contenido web. Es como poner una televisión dentro de tu sala: la televisión muestra su propio canal, pero está dentro de tu casa.

El uso más común es incrustar videos de YouTube. Cuando ves un video en YouTube y le das a "Compartir" y luego a "Insertar", YouTube te da un código `<iframe>` listo para pegar. Lo pegas en tu página y el video aparece ahí, con su propio reproductor, sin que tú tengas que subir ningún archivo de video a tu servidor. Esto tiene una ventaja enorme: YouTube se encarga del almacenamiento, la compresión y la reproducción en todos los dispositivos. Tu página solo muestra la ventana. Por eso casi todos los sitios que muestran videos usan YouTube o plataformas similares en lugar de subir sus propios mp4.

El otro uso clásico es Google Maps. En Google Maps buscas una dirección, le das a "Compartir" y luego a "Insertar un mapa", y te da un `<iframe>` con el mapa interactivo de ese lugar: el visitante puede moverlo, acercarlo y ver las calles sin salir de tu página. Para negocios es oro puro: página de contacto con mapa embebido es el estándar profesional. El visitante ve dónde estás y puede trazar su ruta con un clic.

Ahora los atributos importantes. El `src` es la dirección de la página que quieres incrustar (obligatorio, sin él el `iframe` está vacío). Los atributos `width` y `height` definen el tamaño de la ventana en píxeles. Pero el atributo más importante para la calidad de tu código es `title`: todo `<iframe>` debe llevar un `title` descriptivo, por ejemplo `title="Video de presentación de la academia"` o `title="Mapa de ubicación de la academia"`. ¿Por qué es obligatorio? Porque para una persona ciega que usa lector de pantalla, un `iframe` sin título es una caja misteriosa sin nombre: el lector no sabe qué hay dentro. El `title` es como ponerle un letrero a la ventana: "aquí hay un mapa". Es un requisito de accesibilidad y también una buena práctica que los validadores de código te van a exigir. Nunca publiques un `iframe` sin `title`.

El atributo `sandbox` es tu sistema de seguridad. Un `iframe` muestra contenido de otro sitio, y ese contenido podría intentar hacer cosas que no quieres (como abrir ventanas emergentes o ejecutar código malicioso). Con `sandbox` limitas lo que el `iframe` puede hacer. Si pones solo `sandbox` sin valor, aplicas todas las restricciones: el contenido se muestra pero no puede ejecutar scripts ni abrir ventanas. Puedes aflojar restricciones específicas con valores como `allow-scripts` o `allow-same-origin`, pero la regla es: empieza con el máximo de restricciones y solo permite lo mínimo necesario para que el contenido funcione. Es como recibir visitas en tu casa: les abres la sala, no toda la casa.

El atributo `loading="lazy"` es un regalo para la velocidad de tu página. Significa "carga este `iframe` solo cuando el visitante se acerque a él al hacer scroll". Si tu página tiene un mapa hasta abajo, ¿para qué descargarlo desde el inicio si el visitante quizá ni baja hasta ahí? Con `lazy loading` la página inicial carga rapidísimo y el mapa se carga cuando hace falta. Úsalo siempre en iframes que no están en la parte visible inicial de la página. Es una de esas mejoras invisibles que hacen tu sitio sentirse rápido y profesional.

También existe el atributo `allowfullscreen` (en los iframes de YouTube viene incluido) que permite al visitante ver el video en pantalla completa, y `frameborder` que en HTML5 ya no se usa (los bordes se controlan con CSS). Mención breve para dos etiquetas parientes: `<embed>` y `<object>`. Son las formas antiguas y genéricas de incrustar contenido externo (se usaban mucho para Flash). Hoy casi todo se hace con `<iframe>`, que es más simple y más seguro, pero si las ves en código viejo ya sabes qué son: intentos anteriores de lograr lo mismo.

Errores comunes con iframe: primero y más grave, olvidar el title (caja sin nombre para lectores de pantalla). Segundo, incrustar contenido sin sandbox cuando no confías del todo en la fuente. Tercero, no usar loading="lazy" y que la página tarde en cargar por un mapa que está hasta abajo. Cuarto, poner width y height fijos gigantes que rompen el diseño en celulares (más adelante aprenderás a hacer iframes responsivos con CSS; por ahora usa tamaños razonables). Y quinto, incrustar páginas que prohíben ser incrustadas: algunos sitios bloquean los iframes por seguridad, y en ese caso verás un cuadro vacío; no es tu culpa, es decisión del otro sitio.

Buenas prácticas: title siempre, sandbox cuando el contenido no es de tu total confianza, loading="lazy" en todo lo que esté fuera de la vista inicial, y prefiere proveedores confiables (YouTube, Google Maps, Spotify) para tu contenido embebido. Y un consejo de oro: el iframe muestra la página externa tal como está; si esa página cambia o se cae, tu ventana lo refleja. Para contenido crítico de tu negocio, considera si te conviene depender de un tercero o alojarlo tú.

Cómo se conectan las tres clases

Antes de pasar a los ejemplos, detente un momento a ver el panorama. Las clases 16, 17 y 18 te dieron tres superpoderes distintos: multimedia propia (audio/video), datos organizados (tablas) y contenido ajeno integrado (iframe). En una página real los tres conviven: imagina la página de un curso de la academia. Arriba, un video de presentación con <video> y su poster atractivo. En medio, una tabla con el temario: thead con los títulos (Clase, Tema, Duración), tbody con las 30 clases, tfoot con el total de horas. Abajo, un mapa embebido con <iframe> mostrando dónde son las clases presenciales, más un video de YouTube con testimonios. Una sola página, tres técnicas, resultado profesional. Eso es lo que vas a practicar en los ejemplos y ejercicios que siguen: no piezas sueltas, sino combinaciones reales.

Formatos, compresión y peso: la parte que nadie ve pero todos sienten

Hablemos de un tema poco glamoroso pero decisivo: el peso de tus archivos multimedia. Un video sin comprimir puede pesar cientos de megabytes. Eso significa que un visitante con datos móviles podría gastar una buena parte de su plan solo por abrir tu página, y además tendría que esperar una eternidad a que cargue. La compresión es el arte de hacer los archivos más ligeros sin que se note la pérdida de calidad, y como creador web es tu responsabilidad.

Para audio, la regla práctica es simple: exporta tus mp3 a 128 kbps si es música de fondo o voz, y a 192 kbps si la calidad es importante (por ejemplo, un curso de música). Más de eso

casi nadie lo nota en una bocina de laptop o en audífonos de celular, pero el archivo pesa mucho más. El formato ogg con calidad similar pesa parecido al mp3. El wav no lo uses para web salvo que tengas una razón muy específica: un minuto de wav puede pesar 10 MB, mientras el mismo minuto en mp3 pesa 1 MB.

Para video, el mp4 con códec H.264 es el estándar de la web: buena calidad con archivos razonables. Un video de 720p bien comprimido puede pesar entre 5 y 10 MB por minuto. Si tu video pesa 50 MB por minuto, algo está mal en tu exportación. Antes de subir un video a tu servidor, pásalo por un compresor o por la configuración de exportación de tu editor con destino web. Y recuerda la alternativa inteligente: si el video es largo o muy visto, súbelo a YouTube y embébelo con iframe. YouTube tiene una infraestructura de compresión y distribución que ningún servidor casero puede igualar.

Otro truco de peso: no sirvas el mismo video gigante a todos. Para un video de fondo decorativo, usa una versión corta, en silencio y muy comprimida. Para el contenido principal, la versión de buena calidad. Y usa preload="none" en los videos que no son lo primero que el visitante va a ver. Cada megabyte que no descargas de entrada es velocidad que ganas.

Accesibilidad: que nadie se quede fuera

La accesibilidad no es un extra para después: es parte de hacer bien tu trabajo desde el inicio. Y en esta unidad hay mucho que puedes hacer con poco esfuerzo. Repasemos todo junto para que lo tengas como lista mental.

Para audio y video: agrega subtítulos con <track> siempre que haya diálogo o narración importante. Escribe el texto alternativo dentro de las etiquetas. No dependas solo del color para transmitir información en tus videos (por ejemplo, si en el video dices "lo rojo es peligroso", alguien daltónico puede no distinguirlo: descríbelo con palabras). Si el video tiene información visual clave que no se narra, considera una descripción de audio o al menos un resumen escrito debajo del video.

Para tablas: usa <th> con scope, <caption> descriptivo y la estructura thead/tbody/tfoot. Evita tablas con celdas unidas demasiado complejas, porque los lectores de pantalla las interpretan peor: si tu tabla parece un crucigrama, probablemente conviene dividirla en dos tablas simples. Y nunca uses tablas para acomodar el diseño visual de la página: eso confunde totalmente a los lectores de pantalla, que anuncian "tabla con 3 filas y 4 columnas" cuando en realidad es un menú de navegación.

Para iframes: el title descriptivo es obligatorio, sin excepciones. Además, si el contenido embebido es esencial (por ejemplo, el mapa con tu ubicación), ofrece una alternativa en texto:

la dirección escrita y un enlace a Google Maps. Así, si el iframe falla o el visitante usa un lector de pantalla, la información sigue disponible.

Piensa en la accesibilidad como construir una rampa junto a las escaleras: no le quita nada a quien usa las escaleras, pero le abre la puerta a quien la necesita. Y hay un beneficio extra que pocos mencionan: mucho de lo que ayuda a la accesibilidad también ayuda al SEO, porque los buscadores entienden mejor tu contenido cuando está bien estructurado y descrito.

Rendimiento: páginas rápidas aunque tengan de todo

Una página con tres videos, dos audios y un mapa puede cargar rapidísimo o ser un desastre: la diferencia está en las decisiones que tomes. Aquí tienes la estrategia completa de rendimiento para esta unidad.

Primero, carga diferida: usa `preload="none"` o `preload="metadata"` en audio y video que no sean lo principal, y `loading="lazy"` en iframes fuera de la vista inicial. Esto hace que la página muestre su contenido esencial de inmediato y deje lo pesado para cuando se necesite.

Segundo, el atributo `poster` no solo es bonito: evita que el navegador tenga que procesar el primer cuadro del video para mostrar algo, y te permite mostrar una imagen ligera mientras el video se prepara.

Tercero, cuidado con el autoplay: un video con autoplay (aunque sea muted) empieza a descargar y reproducir desde el primer segundo, consumiendo datos y batería. Úsalo solo cuando el video sea parte central del diseño, no por decorar.

Cuarto, tamaños razonables: no pongas un video de 1920 píxeles de ancho en un reproductor de 640. Sirve versiones acordes al tamaño en que se van a mostrar. Lo mismo con el poster: una imagen de 200 KB es suficiente, no necesitas una foto de 5 MB.

Quinto, limita la cantidad: si tu página necesita diez videos, pregúntate si no conviene una página por video o una lista de enlaces. La mejor optimización es no cargar lo que no se necesita.

Para medir, usa las herramientas de desarrollo de tu navegador (tecla F12, pestaña de red): ahí ves cuánto pesa cada archivo y cuánto tarda. Si tu página tarda más de 3 segundos en mostrar lo esencial, revisa tus archivos multimedia: casi siempre ahí está el problema.

Casos reales: así se usa en páginas de verdad

Cerremos la explicación con casos reales para que veas cómo todo se combina fuera de los ejemplos de libro.

Caso 1: página de un restaurante. Arriba, un video de fondo con autoplay muted loop mostrando el local (corto, comprimido, con poster por si tarda). En la sección de menú, una tabla con thead/tbody: platillos, descripciones y precios, con tfoot para el aviso de "precios con IVA incluido". Abajo, el mapa embebido con la ubicación y loading="lazy". Tres técnicas de esta unidad en una sola página que vende.

Caso 2: página de un curso en línea. Video de presentación con <video>, poster atractivo y subtítulos en español e inglés (dos <track>). Tabla del temario con las 30 clases en el tbody y el total de horas en el tfoot. Video testimonial de YouTube embebido con iframe. Todo lo de la unidad, ordenado como lo vería un alumno real.

Caso 3: portafolio de un músico. Reproductor <audio> con sus canciones en mp3 y ogg. Tabla de próximas fechas de conciertos con thead/tbody/tfoot y rowspan para los festivales de varios días. Mapa embebido del próximo concierto. El músico no necesita saber programar nada avanzado: con esta unidad arma su página completa.

Caso 4: blog de noticias. Videos mp4 para los reportajes con subtítulos (muchas gente lee noticias en silencio en el transporte). Tablas para datos: resultados deportivos, cifras económicas. Iframes para contenido de redes sociales y mapas de los lugares de la noticia. La unidad 6 es, en la práctica, el kit diario de un sitio de noticias.

Fíjate en el patrón: el contenido propio e importante va con <audio> y <video> en tu servidor; los datos estructurados van en tablas semánticas; y el contenido de plataformas gigantes va embebido con iframe. Esa es la arquitectura mental que debes llevarte: cada técnica en su lugar, combinadas con criterio.

Glosario rápido de atributos de la unidad

Cuando llevas un rato programando, lo que más consultas no son las etiquetas (esas te las aprendes rápido) sino los atributos: ¿cómo se llamaba el de los subtítulos? ¿cuál era el de carga diferida? Aquí tienes el glosario completo de esta unidad, para consulta rápida.

Atributos de <audio> y <video>: controls (muestra los botones del reproductor; sin él no hay forma de darle play salvo con JavaScript), src (ruta del archivo, solo cuando no usas <source>), autoplay (empieza solo al cargar; con sonido lo bloquean los navegadores, úsalo solo con muted), muted (empieza en silencio), loop (repite al terminar), preload con valores

none, metadata y auto (controla cuánto se descarga antes del play), width y height (tamaño del reproductor de video en píxeles; en <video> reservan el espacio para que la página no brinque).

Atributos exclusivos de <video>: poster (imagen de portada antes del play; úsalo siempre en videos importantes).

Atributos de <source>: src (ruta del archivo en ese formato) y type (tipo MIME como audio/mpeg, audio/ogg, video/mp4, video/webm; ayuda al navegador a decidir sin descargar).

Atributos de <track>: src (ruta del archivo.vtt), kind (tipo de pista: subtitles es el más usado), srclang (idioma, como es), label (nombre visible en el menú del reproductor).

Atributos de tablas: scope en <th> con valores col y row (dice si el encabezado describe una columna o una fila), colspan (cuántas columnas abarca una celda), rowspan (cuántas filas abarca una celda).

Atributos de <iframe>: src (página a incrustar, obligatorio), width y height (tamaño de la ventana), title (descripción para accesibilidad, obligatorio en la práctica), sandbox (restringe lo que el contenido embebido puede hacer; solo sandbox aplica todas las restricciones), loading con valor lazy (carga diferida cuando el visitante se acerca al hacer scroll), allowfullscreen (permite pantalla completa en videos).

Guárdalos en tu memoria a largo plazo por grupos: "controles de reproducción" (controls, autoplay, muted, loop, preload), "presentación" (poster, width, height, caption), "respaldo" (source, type, track), "estructura de tabla" (thead, tbody, tfoot, scope, colspan, rowspan) y "embebido seguro" (title, sandbox, loading). Agrupados así son mucho más fáciles de recordar que en lista suelta.

Lista de verificación: antes de publicar tu página

Usa esta lista cada vez que termines una página con contenido de esta unidad. Es el mismo tipo de revisión que hacen los profesionales antes de entregar un trabajo.

Reproductores: ¿todos tus <audio> y <video> tienen controls (salvo que los controles con JavaScript a propósito)? ¿ofreces al menos dos formatos con <source> en el contenido importante? ¿tienen texto alternativo? ¿los videos importantes tienen poster? ¿los videos con diálogo tienen subtítulos con <track>? ¿evitaste el autoplay con sonido?

Peso y velocidad: ¿comprimiste tus archivos antes de subirlos? ¿usaste `preload="none"` o `"metadata"` en los archivos que no son lo primero que se ve? ¿los iframes fuera de la vista inicial llevan `loading="lazy"`? ¿el poster no es una imagen gigante?

Tablas: ¿los encabezados usan `<th>` con `scope` y no `<td>`? ¿la tabla está dividida en `thead`, `tbody` y `tfoot` (con `tfoot` antes del `tbody` en el código)? ¿tiene `<caption>` descriptivo? ¿contaste las celdas de cada fila donde usaste `colspan` o `rowspan`? ¿la tabla muestra datos de verdad y no está acomodando el diseño?

Iframes: ¿cada iframe tiene un `title` descriptivo? ¿usaste `sandbox` si la fuente no es de tu total confianza? ¿el contenido embebido esencial tiene alternativa en texto (dirección escrita, enlace)? ¿los tamaños son razonables para celular?

Si respondes que sí a todo, tu página está lista. Si algo falla, ya sabes exactamente dónde mirar: esta unidad te dio las herramientas y esta lista te da el orden para revisarlas.

Preguntas frecuentes de la unidad

¿Puedo poner autoplay con sonido si el video es mío y yo decido? No. El bloqueo no es una sugerencia: los navegadores lo impiden por código. Si lo intentas, el video simplemente no empezará o empezará en silencio según el navegador. Diseña siempre pensando en que el visitante decide cuándo hay sonido.

¿YouTube o video propio, cómo decido? Usa YouTube (con iframe) cuando el video es largo, cuando esperas muchas visitas, o cuando no quieres complicarte con formatos y compresión. Usa `<video>` propio cuando el video es corto y central en tu diseño (como un video de fondo), cuando no quieres la marca de YouTube, o cuando necesitas control total sobre el reproductor.

¿Y si el iframe de un sitio me sale vacío? Algunos sitios bloquean que los muestren dentro de iframes por seguridad. No es un error en tu código: es decisión del otro sitio. Prueba con la página de otro proveedor o enlaza al contenido en lugar de embeberlo.

¿Las tablas se ven feas sin CSS, hice algo mal? No. HTML pone la estructura y el significado; la belleza visual la pone CSS, que verás en unidades posteriores. Una tabla "fea" pero con `thead`, `th` con `scope` y `caption` está perfecta: ya es profesional por dentro, y el estilo llegará.

¿Puedo anidar un iframe dentro de otro? Técnicamente sí, pero casi nunca es buena idea: cada nivel suma carga y complejidad. Si sientes que necesitas iframes anidados, probablemente haya una forma más simple de lograrlo.

¿Necesito permiso para embeber un video de YouTube o un mapa? YouTube y Google Maps ofrecen la opción de insertar justamente para eso: si el autor activó la inserción, puedes usar el código que te dan. Respeta cuando un creador la desactiva y no busques formas de rodearlo.

Ejemplos paso a paso

Ejemplo 1: reproductor de audio con varios formatos

Vamos a crear un reproductor de música que funcione en todos los navegadores. La estrategia: ofrecer el mismo audio en mp3 y en ogg usando `<source>`, para que el navegador elija el que entienda. Además agregamos texto alternativo por si nada funciona.

```
<!-- Reproductor de audio con formatos de respaldo -->
<!-- La etiqueta audio crea el reproductor -->
<!-- controls muestra los botones de play, pausa y volumen -->
<audio controls preload="metadata">
  <!-- Primer intento: formato mp3, el mas universal -->
  <!-- type ayuda al navegador a decidir sin descargar el archivo -->
  <source src="cancion.mp3" type="audio/mpeg">
  <!-- Plan B: formato ogg, libre y de buena calidad -->
  <source src="cancion.ogg" type="audio/ogg">
  <!-- Texto alternativo: solo se ve si nada se pudo reproducir -->
  Tu navegador no soporta audio HTML5. Descarga la cancion aqui.
</audio>
```

Paso a paso: primero abrimos `<audio>` con `controls` para que aparezcan los botones y `preload="metadata"` para que solo descargue datos básicos al inicio (la página carga rápido). Dentro ponemos dos `<source>`: el navegador prueba el mp3 primero; si lo entiende, lo reproduce y ignora el resto. Si no lo entiende, pasa al ogg. Al final, el texto alternativo como red de seguridad. Cerramos con `</audio>`. Fíjate que NO usamos el atributo `src` en la etiqueta `<audio>` cuando usamos `<source>`: son dos formas distintas de decirle dónde está el archivo, y `<source>` es la forma flexible.

Ejemplo 2: video con subtítulos y poster

Ahora un video profesional: con imagen de portada (poster), varios formatos de respaldo y subtítulos en español. Este es el estándar que usan los sitios serios.

```

<!-- Video con poster, formatos de respaldo y subtítulos -->
<!-- width y height reservan el espacio para que la página no brinque -->
<video controls width="640" height="360" poster="portada-curso.jpg"
preload="metadata">
  <!-- Formato principal: mp4, lo reproduce casi todo -->
  <source src="presentacion.mp4" type="video/mp4">
  <!-- Respaldo: webm, formato libre impulsado por Google -->
  <source src="presentacion.webm" type="video/webm">
  <!-- Subtítulos en español en formato WebVTT -->
  <!-- kind="subtitles" indica que es una pista de subtítulos -->
  <!-- srclang="es" dice que están en español -->
  <!-- label es el nombre que ve el visitante en el menú -->
  <track src="subtitulos-es.vtt" kind="subtitles" srclang="es" label="Español">
  <!-- Texto alternativo si el video no se puede reproducir -->
  Tu navegador no soporta video HTML5.
</video>

```

Paso a paso: abrimos <video> con controls, tamaño 640x360, poster con una imagen atractiva que invita a darle play, y preload="metadata". Los dos <source> cubren mp4 y webm. La etiqueta <track> agrega los subtítulos: el archivo subtitulos-es.vtt contiene los textos con sus tiempos. Nota que <track> no lleva etiqueta de cierre: es una etiqueta vacía como . El archivo.vtt se ve así por dentro:

WEBVTT

00:00.000 --> 00:04.000

Bienvenido al curso de HTML5 de Lucas EGA Academy.

00:04.000 --> 00:09.000

En esta lección aprenderás a usar audio y video nativos.

Cada bloque lleva el tiempo de inicio y fin, y debajo el texto del subtítulo. Simple y poderoso.

Ejemplo 3: tabla de horarios con thead, tbody, tfoot y celdas unidas

Vamos a armar la tabla de horarios de un grupo: encabezado con thead, cuerpo con tbody, pie con tfoot, y celdas unidas con colspan para el recreo y rowspan para una materia de dos horas. Dibuja esto en papel primero si puedes.

```

<!-- Tabla de horarios profesional -->
<table>
  <!-- Título oficial de la tabla, ligado a ella -->

```

```

<caption>Horario de clases - Grupo A</caption>
<!-- Encabezado: titulos de las columnas -->
<thead>
  <tr>
    <!-- scope="col": cada th describe su columna -->
    <th scope="col">Hora</th>
    <th scope="col">Lunes</th>
    <th scope="col">Martes</th>
    <th scope="col">Miercoles</th>
  </tr>
</thead>
<!-- El pie se escribe ANTES del cuerpo, pero se muestra al final -->
<tfoot>
  <tr>
    <!-- colspan="4": esta celda abarca las 4 columnas -->
    <td colspan="4">Total de horas por semana: 20</td>
  </tr>
</tfoot>
<!-- Cuerpo: los datos fila por fila -->
<tbody>
  <tr>
    <!-- scope="row": este th describe su fila -->
    <th scope="row">8:00 - 9:00</th>
    <td>HTML5</td>
    <td>CSS3</td>
    <td>HTML5</td>
  </tr>
  <tr>
    <th scope="row">9:00 - 10:00</th>
    <!-- colspan="2": el recreo abarca lunes y martes -->
    <!-- esta fila lleva 3 celdas en vez de 4, porque una cuenta doble -->
    <td colspan="2">Recreo</td>
    <td>CSS3</td>
  </tr>
  <tr>
    <th scope="row">10:00 - 11:00</th>
    <!-- rowspan="2": JavaScript dura dos horas el lunes -->
    <td rowspan="2">JavaScript</td>
    <td>Python</td>
    <td>Python</td>
  </tr>
  <tr>
    <th scope="row">11:00 - 12:00</th>
    <!-- aqui NO repetimos JavaScript: la celda de arriba ya ocupa este lugar -->

```

```

        <td>SQL</td>
        <td>SQL</td>
    </tr>
</tbody>
</table>

```

Paso a paso: el <caption> nombra la tabla. El thead tiene una fila con cuatro <th> de scope="col". El tfoot (antes del tbody en el código) tiene una fila con una sola celda de colspan="4" que abarca todo el ancho para el total. En el tbody, la fila del recreo usa colspan="2" para unir lunes y martes, y por eso esa fila solo declara 3 celdas. La celda de JavaScript usa rowspan="2" para durar dos horas, y la fila siguiente NO incluye celda para el lunes porque ese espacio ya está ocupado. Cuenta las celdas con cuidado: es el secreto de las tablas con celdas unidas.

Ejemplo 4: página con video de YouTube y mapa embebidos

Ahora una página de contacto completa que combina iframe de YouTube y de Google Maps. Fíjate en el title de cada iframe y en loading="lazy" en el mapa.

```

<!-- Pagina de contacto con contenido embebido -->
<h1>Visitanos</h1>
<p>Mira nuestro video de presentacion y encuentra nuestra ubicacion en el mapa.</p>

<!-- Video de YouTube embebido -->
<!-- El codigo lo obtienes en YouTube: Compartir > Insertar -->
<!-- title obligatorio: describe el contenido para lectores de pantalla -->
<!-- allowfullscreen permite ver el video en pantalla completa -->
<iframe
                                width="560"
                                height="315"
    src="https://www.youtube.com/embed/CODIGO_DEL_VIDEO"  title="Video  de presentacion
de Lucas EGA Academy" allowfullscreen>
</iframe>

<h2>Nuestra ubicacion</h2>

<!-- Mapa de Google Maps embebido -->
<!-- El codigo lo obtienes en Google Maps: Compartir > Insertar un mapa -->
<!-- loading="lazy": el mapa carga solo cuando el visitante baja hasta el -->
<iframe  src="https://www.google.com/maps/embed?pb=DIRECCION_DEL_MAPA"  width="600"
height="450" title="Mapa de ubicacion de Lucas EGA Academy" loading="lazy">
</iframe>

<p>Te esperamos de lunes a viernes de 9:00 a 18:00.</p>

```


Paso a paso: el encabezado y el párrafo presentan la página. El primer iframe muestra el video de YouTube: pegas el código que te da YouTube y solo te aseguras de que tenga un title descriptivo. El segundo iframe muestra el mapa con loading="lazy" para no frenar la carga inicial. Ambos iframes llevan title, y el mapa además lleva lazy loading porque está más abajo en la página. Cuando publiques tu página real, reemplaza CODIGO_DEL_VIDEO y DIRECCION_DEL_MAPA con los códigos que te den YouTube y Google Maps. Así de simple es tener video y mapa profesionales sin subir ni un solo archivo pesado a tu servidor.

Ejemplo 5: página de curso que combina video, tabla e iframe

El ejemplo final junta las tres clases en una página real: la presentación de un curso con video propio, tabla del temario y mapa de la sede presencial. Así se ve la unidad trabajando en equipo.

```
<!-- Pagina de un curso: video + tabla + mapa -->
<h1>Curso de HTML5 desde cero</h1>

<!-- Video de presentacion con todo lo aprendido -->
<!-- poster atractivo, dos formatos, subtítulos en español -->
<video controls width="640" height="360" poster="curso-portada.jpg"
preload="metadata">
  <source src="curso-presentacion.mp4" type="video/mp4">
  <source src="curso-presentacion.webm" type="video/webm">
  <track src="curso-subtitulos-es.vtt" kind="subtitles" srclang="es" label="Espanol">
  Tu navegador no soporta video HTML5.
</video>

<!-- Temario en tabla profesional -->
<h2>Temario del curso</h2>
<table>
  <caption>Temario completo del curso de HTML5</caption>
  <thead>
    <tr>
      <th scope="col">Clase</th>
      <th scope="col">Tema</th>
      <th scope="col">Duracion</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <!-- colspan="2": el total abarca clase y tema -->
      <td colspan="2">Duracion total del curso</td>
      <td>6 horas</td>
    </tr>
  </tfoot>
</table>
```

```

    </tr>
</tfoot>
<tbody>
  <tr>
    <th scope="row">1</th>
    <td>Que es HTML5</td>
    <td>12 min</td>
  </tr>
  <tr>
    <th scope="row">2</th>
    <td>Estructura de una web</td>
    <td>15 min</td>
  </tr>
  <tr>
    <!-- rowspan="2": el proyecto final abarca dos filas -->
    <th scope="row" rowspan="2">3 y 4</th>
    <td>Proyecto final parte 1</td>
    <td>20 min</td>
  </tr>
  <tr>
    <td>Proyecto final parte 2</td>
    <td>20 min</td>
  </tr>
</tbody>
</table>

<!-- Ubicacion de las clases presenciales -->
<h2>Donde son las clases presenciales</h2>
<iframe src="https://www.google.com/maps/embed?pb=DIRECCION_DEL_MAPA" width="600"
height="450" title="Mapa de la sede del curso de HTML5" loading="lazy">
</iframe>

```

Paso a paso: el <h1> presenta el curso. El <video> trae poster, dos formatos y subtítulos: la presentación oficial. La tabla organiza el temario con caption, thead con scope="col", tbody con scope="row" en los números de clase, y un tfoot donde colspan="2" une las dos primeras columnas para el total. Fíjate en el rowspan="2" de las clases 3 y 4: la celda del número abarca dos filas y la segunda fila ya no la repite. Al final, el mapa embebido con title y loading="lazy" muestra la sede. Una página completa, profesional y accesible, construida solo con lo que aprendiste en esta unidad.

Ejemplo 6: video de fondo con autoplay en silencio

Muchas portadas modernas usan un video de fondo que se mueve solo, sin sonido, detrás del título del sitio. Es el único caso donde el autoplay es correcto: combinado con muted y loop. Vamos a construirlo bien.

```
<!-- Video de fondo: se mueve solo, en silencio y en repeticion -->
<!-- autoplay + muted: el unico autoplay que los navegadores permiten -->
<!-- loop: al terminar vuelve a empezar, movimiento continuo -->
<!-- playsinline: en iPhone evita que el video se abra en pantalla completa solo -->
<video autoplay muted loop playsinline poster="fondo-portada.jpg" preload="auto">
  <source src="fondo.mp4" type="video/mp4">
  <source src="fondo.webm" type="video/webm">
</video>
<h1>Bienvenido a Lucas EGA Academy</h1>
<p>Aprende tecnologia desde cero, gratis y en espanol.</p>
```

Paso a paso: el <video> lleva autoplay muted loop juntos: empieza solo, en silencio (por eso el navegador lo permite) y se repite sin parar. El atributo playsinline es importante en iPhone: sin él, iOS abre cualquier video con autoplay en pantalla completa automáticamente, arruinando el efecto de fondo. El poster muestra una imagen mientras el video se prepara. Los dos <source> dan respaldo de formatos. Nota que NO lleva controls: un video de fondo no necesita botones, es decoración en movimiento. Y nota que no lleva texto alternativo esencial porque es decorativo; el contenido real (el título y el párrafo) está en HTML normal, accesible para todos.

Tres reglas para videos de fondo: primero, que sea corto y ligero (unos segundos en loop, bien comprimido), porque se descarga al abrir la página. Segundo, que no distraiga del contenido: el texto encima debe leerse perfecto; si el video es muy movido, se baja su intensidad con CSS más adelante. Tercero, nunca pongas información importante solo en el video de fondo: si alguien no lo ve (conexión lenta, lector de pantalla), el mensaje debe estar en el HTML, como en este ejemplo.

Práctica

Ejercicio 1: reproductor de podcast con respaldo de formatos

Enunciado: crea un reproductor de audio para el primer episodio de un podcast llamado "Mi primer podcast". Debe mostrar los controles, ofrecer el audio en mp3 y ogg como respaldo, descargar solo los metadatos al inicio para no frenar la página, e incluir texto alternativo.

Solución paso a paso:

Paso 1: abre la etiqueta <audio> con el atributo controls para que el visitante tenga los botones de reproducción.

Paso 2: agrega preload="metadata" para que el navegador solo descargue datos básicos (como la duración) hasta que el visitante le dé play. Así la página carga rápido.

Paso 3: dentro, agrega el primer <source> con el mp3, que es el formato más universal: <source src="episodio1.mp3" type="audio/mpeg">.

Paso 4: agrega el segundo <source> con el ogg como plan B: <source src="episodio1.ogg" type="audio/ogg">. El navegador lo usará solo si no entiende el mp3.

Paso 5: escribe el texto alternativo para el caso extremo en que nada se reproduzca.

Paso 6: cierra la etiqueta </audio>.

Código final:

```
<!-- Reproductor del episodio 1 del podcast -->
<audio controls preload="metadata">
  <source src="episodio1.mp3" type="audio/mpeg">
  <source src="episodio1.ogg" type="audio/ogg">
  Tu navegador no soporta audio HTML5. Puedes descargar el episodio aquí.
</audio>
```

Por qué funciona: el navegador intenta los formatos en orden y se queda con el primero que entiende. El atributo type le permite decidir sin descargar nada primero. Y si un visitante usa un navegador muy antiguo, al menos ve el mensaje alternativo.

Ejercicio 2: video de bienvenida con poster y subtítulos

Enunciado: en la página principal de un curso, coloca un video de bienvenida de 640 por 360 píxeles. Debe tener imagen de portada, controles, formatos mp4 y webm, subtítulos en español, y empezar en silencio solo si decides usar autoplay (explica tu decisión).

Solución paso a paso:

Paso 1: abre `<video>` con controls, width="640", height="360" y poster="bienvenida.jpg". El poster muestra una imagen atractiva antes del play.

Paso 2: decide sobre autoplay. Como el video tiene sonido (una bienvenida hablada), NO uses autoplay con sonido porque el navegador lo bloquearía y además molestaría al visitante. La decisión correcta: sin autoplay, el visitante le da play cuando quiera. Si fuera un video de fondo decorativo sin sonido importante, sí podrías usar autoplay muted loop.

Paso 3: agrega los dos `<source>`: presentación.mp4 con type="video/mp4" y presentación.webm con type="video/webm".

Paso 4: agrega la pista de subtítulos: `<track src="bienvenida-es.vtt" kind="subtitles" srclang="es" label="Español">`.

Paso 5: agrega texto alternativo y cierra `</video>`.

Código final:

```
<!-- Video de bienvenida del curso -->
<video      controls      width="640"      height="360"      poster="bienvenida.jpg"
preload="metadata">
  <source src="presentacion.mp4" type="video/mp4">
  <source src="presentacion.webm" type="video/webm">
  <track src="bienvenida-es.vtt" kind="subtitles" srclang="es" label="Español">
  Tu navegador no soporta video HTML5.
</video>
```

Por qué no lleva autoplay: el video tiene narración importante y el autoplay con sonido está bloqueado por los navegadores. Forzarlo sería pelear contra el navegador y molestar al visitante. El visitante decide cuándo empezar: esa es la experiencia correcta.

Ejercicio 3: tabla de precios con encabezados, pie y celda unida

Enunciado: crea la tabla de precios de una academia con tres planes (Básico, Pro, Premium) y tres características (Cursos, Certificado, Tutorías). Usa thead, tbody y tfoot, `<th>` con scope, `<caption>`, y une con colspan la celda del pie que dice "Todos los planes incluyen acceso de por vida".

Solución paso a paso:

Paso 1: abre `<table>` y pon el `<caption>` con el título: "Planes y precios de la academia".

Paso 2: crea el thead con una fila de encabezados: una celda vacía para la esquina (o la palabra "Plan") y luego Básico, Pro, Premium. Los encabezados de columna llevan `scope="col"`.

Paso 3: escribe el tfoot ANTES del tbody (orden correcto del código). Una fila con una sola celda `<td colspan="4">` con el mensaje de acceso de por vida. Son 4 columnas en total (característica + 3 planes), así que `colspan="4"`.

Paso 4: en el tbody, cada fila empieza con un `<th scope="row">` que nombra la característica (Cursos, Certificado, Tutorías), seguido de tres `<td>` con los valores de cada plan.

Paso 5: cierra tbody y table.

Código final:

```

<table>
  <caption>Planes y precios de la academia</caption>
  <thead>
    <tr>
      <th scope="col">Caracteristica</th>
      <th scope="col">Basico</th>
      <th scope="col">Pro</th>
      <th scope="col">Premium</th>
    </tr>
  </thead>
  <tfoot>
    <tr>
      <td colspan="4">Todos los planes incluyen acceso de por vida</td>
    </tr>
  </tfoot>
  <tbody>
    <tr>
      <th scope="row">Cursos</th>
      <td>5</td>
      <td>15</td>
      <td>22</td>
    </tr>
    <tr>
      <th scope="row">Certificado</th>
      <td>No</td>
      <td>Si</td>
      <td>Si</td>
    </tr>
    <tr>
      <th scope="row">Tutorias</th>
      <td>No</td>
      <td>No</td>
      <td>Si</td>
    </tr>
  </tbody>
</table>

```

Por qué está bien hecha: tiene título ligado con caption, encabezados semánticos con scope para accesibilidad, estructura en tres partes, y el pie usa colspan para abarcar todo el ancho con un mensaje general. Esta tabla la entiende igual un visitante que la ve y un lector de pantalla que la escucha.

Ejercicio 4: página de evento con video de YouTube y mapa

Enunciado: crea una página para un taller gratuito de HTML5. Debe incluir un video promocional de YouTube embebido y el mapa de Google Maps con la sede del evento. Ambos iframes deben ser accesibles y el mapa no debe frenar la carga inicial.

Solución paso a paso:

Paso 1: escribe el encabezado <h1> con el nombre del evento y un párrafo con la fecha y la invitación.

Paso 2: pega el iframe del video de YouTube (código de Compartir > Insertar). Verifica que tenga title="Video promocional del taller de HTML5" y allowfullscreen.

Paso 3: agrega un <h2> para la sección de ubicación.

Paso 4: pega el iframe del mapa de Google Maps (código de Compartir > Insertar un mapa). Agrega title="Mapa de la sede del taller" y loading="lazy" para que cargue solo cuando el visitante baje hasta él.

Paso 5: cierra con un párrafo de despedida e invitación.

Código final:

```
<h1>Taller gratuito de HTML5</h1>
<p>Este sabado a las 10:00. Mira el video y apartate tu lugar.</p>

<iframe                                width="560"                                height="315"
src="https://www.youtube.com/embed/CODIGO_DEL_VIDEO"  title="Video  promocional del
taller de HTML5" allowfullscreen>
</iframe>

<h2>Como llegar</h2>

<iframe  src="https://www.google.com/maps/embed?pb=DIRECCION_DEL_MAPA"  width="600"
height="450" title="Mapa de la sede del taller" loading="lazy">
</iframe>

<p>Entrada libre. Te esperamos.</p>
```

Por qué está bien hecha: cada iframe tiene su title descriptivo (accesibilidad), el video permite pantalla completa, y el mapa usa lazy loading para no castigar la velocidad inicial. Recuerda reemplazar los códigos de ejemplo por los reales que te den YouTube y Google Maps.

Ejercicio 5: detecta y corrige los errores

Enunciado: el siguiente código tiene 5 errores relacionados con lo aprendido en esta unidad. Encuéntralos y escribe la versión corregida.

Código con errores:

```
<video autoplay src="fiesta.mp4">
</video>

<table>
  <tr>
    <td>Nombre</td>
    <td>Edad</td>
  </tr>
  <tr>
    <td>Ana</td>
    <td>25</td>
  </tr>
</table>

<iframe src="https://www.youtube.com/embed/abc123" width="560" height="315">
</iframe>
```

Solución paso a paso:

Error 1: el video usa autoplay SIN muted y SIN controls. El autoplay con sonido lo bloquean los navegadores, y sin controls el visitante ni siquiera puede darle play manualmente. Corrección: quitar autoplay (o usar autoplay muted si es video de fondo) y agregar controls.

Error 2: el video no tiene texto alternativo ni formatos de respaldo. Corrección: agregar <source> y texto alternativo.

Error 3: la tabla usa <td> para los encabezados (Nombre, Edad) en lugar de <th>. Corrección: cambiarlos a <th scope="col">.

Error 4: la tabla no tiene thead/tbody ni caption. Corrección: estructurarla con caption, thead y tbody.

Error 5: el iframe no tiene title. Corrección: agregar title="..." descriptivo.

Código corregido:

```

<!-- Video corregido: con controles, sin autoplay con sonido, con respaldo -->
<video controls width="640" height="360" poster="fiesta.jpg" preload="metadata">
  <source src="fiesta.mp4" type="video/mp4">
  <source src="fiesta.webm" type="video/webm">
  Tu navegador no soporta video HTML5.
</video>

<!-- Tabla corregida: con caption, thead, tbody y th con scope -->
<table>
  <caption>Lista de participantes</caption>
  <thead>
    <tr>
      <th scope="col">Nombre</th>
      <th scope="col">Edad</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>Ana</td>
      <td>25</td>
    </tr>
  </tbody>
</table>

<!-- Iframe corregido: con title obligatorio -->
<iframe src="https://www.youtube.com/embed/abc123" width="560" height="315"
title="Video de la fiesta de graduacion" allowfullscreen>
</iframe>

```

Lección del ejercicio: los errores más comunes son justo los que corregiste aquí: autoplay con sonido, encabezados con <td> en vez de <th>, tablas sin estructura y iframes sin title. Si evitas estos cinco, tu código ya está por encima del promedio.

Ejercicio 6: página de podcast con tres episodios

Enunciado: crea la página de un podcast llamado "Código y Café" con sus tres primeros episodios. Cada episodio debe tener su título, una breve descripción y su propio reproductor de audio con formatos mp3 y ogg. La página debe cargar rápido aunque tenga tres reproductores.

Solución paso a paso:

Paso 1: escribe el encabezado <h1> con el nombre del podcast y un párrafo de presentación. Esto le da contexto a la página antes de los reproductores.

Paso 2: para cada episodio, escribe un <h2> con el título del episodio y un <p> con su descripción. Esto no es decoración: le dice al visitante de qué trata cada audio antes de darle play, y ayuda a los buscadores a entender tu contenido.

Paso 3: debajo de cada descripción coloca un <audio controls preload="none">. ¿Por qué preload="none" y no "metadata"? Porque hay tres audios en la misma página: si cada uno descarga sus metadatos al abrir la página, son tres peticiones innecesarias para audios que quizá el visitante ni escuche. Con "none" no se descarga nada hasta que el visitante le dé play a alguno. Es la decisión de rendimiento correcta cuando hay varios archivos multimedia juntos.

Paso 4: dentro de cada <audio>, pon los dos <source>: primero el mp3 con type="audio/mpeg" y luego el ogg con type="audio/ogg" como respaldo.

Paso 5: agrega el texto alternativo en cada reproductor y cierra las etiquetas.

Paso 6: repite la estructura para los tres episodios. La repetición es intencional: cada episodio es independiente y el visitante elige cuál escuchar.

Código final:

```

<h1>Codigo y Cafe</h1>
<p>El podcast donde hablamos de programacion con una buena taza de cafe. Nuevos episodios cada lunes.</p>

<h2>Episodio 1: Mi primera linea de codigo</h2>
<p>Contamos como empezamos a programar y que nos hubiera gustado saber antes. Duracion: 25 minutos.</p>
<audio controls preload="none">
  <source src="episodio1.mp3" type="audio/mpeg">
  <source src="episodio1.ogg" type="audio/ogg">
  Tu navegador no soporta audio HTML5.
</audio>

<h2>Episodio 2: HTML sin miedo</h2>
<p>Desmitificamos HTML: no es programacion como tal, es estructura, y cualquiera puede aprenderlo. Duracion: 30 minutos.</p>
<audio controls preload="none">
  <source src="episodio2.mp3" type="audio/mpeg">
  <source src="episodio2.ogg" type="audio/ogg">
  Tu navegador no soporta audio HTML5.
</audio>

<h2>Episodio 3: Tu primera pagina en una tarde</h2>
<p>Guia practica para publicar tu primera pagina web hoy mismo, sin gastar un peso. Duracion: 28 minutos.</p>
<audio controls preload="none">
  <source src="episodio3.mp3" type="audio/mpeg">
  <source src="episodio3.ogg" type="audio/ogg">
  Tu navegador no soporta audio HTML5.
</audio>

```

Por qué está bien hecha: cada episodio tiene título y descripción (contenido real, no solo reproductores), cada audio ofrece dos formatos de respaldo, y preload="none" evita que la página descargue tres archivos que quizá nadie escuche. Si en el futuro agregas diez episodios más, esta misma estructura escala sin cambios: solo repites el bloque.

Variante para pensar: si en lugar de tres episodios tuvieras treinta, ¿pondrías los treinta reproductores en una sola página? Probablemente no: mejor una página por episodio o una lista de enlaces a páginas individuales. La mejor optimización, recuerda, es no cargar lo que no se necesita.

Ejercicio 7: tabla de torneo deportivo con video embebido

Enunciado: crea la página de resultados de un torneo relámpago de fútbol con 4 equipos. Debe incluir una tabla con thead, tbody y tfoot, encabezados con scope, caption, una celda unida con colspan para el campeón y una con rowspan para el equipo con dos partidos el mismo día. Además, embebe con iframe el video de YouTube con los mejores goles, accesible y sin frenar la carga.

Solución paso a paso:

Paso 1: escribe el <h1> con el nombre del torneo y un párrafo de introducción. Presenta el contenido antes de los datos.

Paso 2: abre la <table> y pon el <caption>: "Resultados del torneo relampago". El caption le dice a todo el mundo (incluidos los lectores de pantalla) de qué trata la tabla.

Paso 3: crea el thead con una fila de encabezados: Equipo, Partidos jugados, Goles a favor. Cada <th> lleva scope="col" porque describe su columna.

Paso 4: escribe el tfoot ANTES del tbody, con una fila cuya única celda sea <td colspan="3"> con el texto "Campeon del torneo: Halcones". Son 3 columnas, así que colspan="3" abarca todo el ancho.

Paso 5: en el tbody escribe las filas de datos. Cada fila empieza con <th scope="row"> con el nombre del equipo (describe su fila) y luego dos <td> con los números.

Paso 6: para el equipo "Tigres", que jugó dos partidos el mismo día, usa rowspan="2" en su <th> para que su nombre abarque dos filas, y escribe las dos filas de sus partidos. La segunda fila no repite el nombre del equipo porque la celda unida ya ocupa ese lugar. Cuenta con cuidado: la fila del rowspan lleva el <th> más sus <td>, y la fila siguiente solo sus <td>.

Paso 7: después de la tabla, agrega un <h2> para la sección de video y pega el iframe de YouTube con title="Mejores goles del torneo relampago", allowfullscreen y loading="lazy" (el video está abajo en la página, así que lazy loading es lo correcto).

Código final:

```
<h1>Torneo Relampago de Futbol</h1>
<p>Resultados finales del torneo jugado este fin de semana.</p>

<table>
  <caption>Resultados del torneo relampago</caption>
  <thead>
    <tr>
```

```

        <th scope="col">Equipo</th>
        <th scope="col">Partidos jugados</th>
        <th scope="col">Goles a favor</th>
    </tr>
</thead>
<tfoot>
    <tr>
        <td colspan="3">Campeon del torneo: Halcones</td>
    </tr>
</tfoot>
<tbody>
    <tr>
        <th scope="row">Halcones</th>
        <td>3</td>
        <td>9</td>
    </tr>
    <tr>
        <!-- rowspan="2": Tigres jugo dos partidos, su nombre abarca dos filas -->
        <th scope="row" rowspan="2">Tigres</th>
        <td>1 (sabado)</td>
        <td>2</td>
    </tr>
    <tr>
        <!-- aqui no se repite Tigres: la celda de arriba ya ocupa ese lugar -->
        <td>1 (domingo)</td>
        <td>3</td>
    </tr>
    <tr>
        <th scope="row">Lobos</th>
        <td>2</td>
        <td>4</td>
    </tr>
</tbody>
</table>

<h2>Los mejores goles</h2>
<iframe width="560" height="315" src="https://www.youtube.com/embed/CODIGO_DEL_VIDEO"
title="Mejores goles del torneo relampago" loading="lazy" allowfullscreen>
</iframe>

```

Por qué está bien hecha: la tabla tiene estructura completa (caption, thead, tfoot antes del tbody, tbody), encabezados con scope en ambas direcciones, colspan para el anuncio del campeón a todo lo ancho y rowspan para el equipo con doble jornada, con el conteo de celdas correcto en cada fila. El iframe del video lleva title descriptivo, permite pantalla completa y usa

lazy loading porque está al final de la página. Es una página de resultados deportivos de nivel profesional hecha solo con HTML.

Resumen de la unidad

Antes de la lista final, hagamos un repaso mental de lo que lograste. Al empezar esta unidad, tus páginas eran documentos estáticos: texto, imágenes y enlaces. Al terminarla, sabes ponerles sonido y movimiento con `<audio>` y `<video>` nativos, organizar datos como un profesional con tablas estructuradas, y traer contenido del resto del internet a tu página con `<iframe>`. Esas tres habilidades, combinadas, son las que convierten una página simple en una experiencia completa.

Lo más valioso no son las etiquetas aisladas sino el criterio para usarlas: cuándo conviene subir tu propio video y cuándo embeberlo de YouTube, cuándo un autoplay en silencio suma y cuándo molesta, cuándo una tabla necesita celdas unidas y cuándo conviene partirla en dos, cuándo un iframe necesita sandbox y cuándo basta con un title bien escrito. Ese criterio es lo que separa a quien copia código de quien construye páginas de verdad, y esta unidad te lo dio con ejemplos, errores comunes y casos reales.

Guarda bien la lista de verificación y el glosario de atributos: los vas a consultar muchas veces en los próximos proyectos, y cada vez los necesitarás menos. En la siguiente unidad seguirás sumando piezas, y todo lo de aquí (el video de presentación, la tabla del temario, el mapa de contacto) va a reaparecer combinado con lo nuevo. Así se construye un desarrollador: capa sobre capa, sin saltarse ninguna.

Puntos clave de la unidad:

- Con `<audio controls>` y `<video controls>` reproduces música y video de forma nativa en HTML5, sin plugins como Flash.
- Los formatos más universales son mp3 para audio y mp4 para video; ogg, wav, webm y ogv son alternativas válidas.
- La etiqueta `<source>` ofrece varios formatos como respaldo: el navegador usa el primero que entiende, como un plan B automático.
- Pon siempre texto alternativo dentro de `<audio>` y `<video>` para los navegadores que no puedan reproducir nada.
- El atributo autoplay con sonido está bloqueado por los navegadores: úsalo solo combinado con muted, y solo cuando aporte valor.

- Los atributos loop, muted y preload controlan la repetición, el silencio inicial y cuánto se descarga antes del play.
- El atributo poster en <video> muestra una imagen de portada atractiva antes de la reproducción.
- La etiqueta <track> agrega subtítulos en formato WebVTT: accesibilidad para personas con discapacidad auditiva y para quienes ven video en silencio.
- Divide tus tablas en thead (encabezado), tbody (cuerpo) y tfoot (pie); el tfoot se escribe antes del tbody en el código.
- Usa <th> con scope="col" o scope="row" en los encabezados y <caption> como título oficial de la tabla.
- Con colspan unes celdas horizontalmente y con rowspan verticalmente, como combinar celdas en Excel; ajusta el número de celdas de cada fila.
- Con <iframe src="..."> incrustas otra página dentro de la tuya: videos de YouTube, mapas de Google Maps y más.
- Todo iframe debe llevar title descriptivo por accesibilidad, sandbox si la fuente no es de total confianza, y loading="lazy" cuando está fuera de la vista inicial.
- Las etiquetas <embed> y <object> son formas antiguas de incrustar contenido; hoy se prefiere <iframe>.
- Errores que ya no vas a cometer: autoplay con sonido, tablas sin <th>, celdas mal contadas con colspan/rowspan, e iframes sin title.
- Comprime tus archivos antes de subirlos: mp3 a 128 o 192 kbps y mp4 con H.264 bien exportado; cada megabyte ahorrado es velocidad ganada.
- Usa preload="none" o "metadata" cuando haya varios archivos multimedia en la página, y loading="lazy" en iframes fuera de la vista inicial.
- La accesibilidad es parte del trabajo: subtítulos, texto alternativo, scope en encabezados, caption en tablas y title en iframes abren tu página a todos.
- Arquitectura mental de la unidad: contenido propio importante con <audio> y <video> en tu servidor, datos estructurados en tablas semánticas, y contenido de plataformas gigantes embebido con <iframe>.
- Antes de publicar, pasa la lista de verificación: controles, formatos de respaldo, poster, subtítulos, peso, estructura de tabla, conteo de celdas y titles en iframes.

Felicidades: terminaste la unidad 6. Tus páginas ya reproducen audio y video, presentan datos en tablas profesionales e integran contenido de YouTube y Google Maps. En la siguiente unidad seguirás construyendo sobre esta base. Nos vemos ahí.

UNIDAD 7: Gráficos y listas especializadas

Bienvenido a la unidad 7 del curso completo de HTML5 de Lucas EGA Academy. Si llegaste hasta aquí es porque ya dominas lo esencial: sabes estructurar una página con las etiquetas semánticas, dar formato al texto, crear enlaces que conectan tu sitio con el mundo, mostrar imágenes, organizar datos en tablas, construir formularios que recogen información de tus visitantes y hasta incrustar audio y video. Tienes una base sólida, de esas que se notan cuando alguien abre tu página y todo se ve ordenado, claro y profesional. Ahora toca dar el siguiente salto: vas a aprender a crear gráficos desde cero con tu propio código y a organizar información con listas especializadas que la mayoría de los principiantes ni siquiera sabe que existen.

Esta unidad cubre las clases 19, 20 y 21 del curso. La clase 19 te presenta a canvas, que es básicamente una pizarra digital dentro de tu página donde puedes dibujar con JavaScript: rectángulos, líneas, círculos, texto y todo lo que tu imaginación quiera poner ahí. La clase 20 te enseña SVG, una forma de dibujar gráficos vectoriales directamente en HTML, perfectos para logos e iconos que se ven nítidos en cualquier tamaño de pantalla, desde un celular pequeño hasta un monitor gigante. Y la clase 21 te muestra las listas de definición, una herramienta elegante para crear glosarios, preguntas frecuentes y cualquier contenido donde un término necesita su explicación al lado.

¿Por qué importa tanto esta unidad? Piensa en las páginas web que más te gustan, las que visitas todos los días. Casi ninguna es puro texto. Todas tienen gráficos, iconos, logotipos, diagramas o ilustraciones que hacen la experiencia más clara y más bonita. Antes de HTML5, si querías un gráfico en tu página tenías dos opciones: usar una imagen hecha en otro programa, o sufrir con tecnologías complicadas que pedían programas extra instalados en la computadora del visitante. HTML5 cambió eso para siempre: ahora el navegador trae de fábrica las herramientas para dibujar. Con canvas y SVG puedes generar gráficos al vuelo, animarlos, hacerlos interactivos y adaptarlos a cada pantalla sin depender de archivos externos pesados.

Además, esta unidad encaja en el curso como la pieza que convierte tus páginas de documentos estáticos en experiencias visuales. En las unidades anteriores aprendiste a presentar información; aquí aprendes a ilustrarla. Un gráfico bien puesto explica en tres segundos lo que un párrafo tarda un minuto en decir. Una lista de definición bien armada convierte una pared de texto aburrida en un glosario que da gusto leer. Y cuando combines lo que aprendas aquí con JavaScript más adelante, vas a poder crear desde gráficas de barras dinámicas hasta pequeños juegos completos corriendo dentro del navegador.

Otro punto importante: canvas y SVG no son caprichos de diseñador, son habilidades que se piden en el mundo laboral. Las empresas necesitan gente que sepa hacer tableros de datos con gráficas, iconos personalizados que no se pixelen en pantallas de alta resolución, y visualizaciones interactivas. Dominar estas etiquetas te pone por delante de muchísima gente que se quedó en lo básico de HTML y nunca se atrevió a dibujar con código. Y lo mejor es que no necesitas ser un artista: aquí el dibujo se hace con instrucciones precisas, con números y coordenadas, no con pulso de pintor.

La unidad está organizada para que avances paso a paso. Primero viene la explicación detallada, donde desmenuzamos cada etiqueta, cada atributo y cada concepto con analogías que te van a hacer decir 'ah, ya entendí'. Después tienes ejemplos paso a paso, código completo y comentado que puedes copiar, pegar y modificar para experimentar. Luego la sección de práctica, con cinco ejercicios que van de lo sencillo a lo retador, cada uno con su solución explicada para que compares tu trabajo. Y cerramos con un resumen de los puntos clave para que repases rápido antes de seguir a la siguiente unidad.

Un consejo antes de empezar: esta es una de esas unidades donde leer no basta. Abre tu editor, escribe el código de los ejemplos y míralo funcionar en el navegador. Cambia los números, cambia los colores, rompe cosas a propósito y luego arréglalas. Dibujar con código se aprende dibujando, igual que andar en bici se aprende pedaleando. Nadie aprendió canvas solo leyendo teoría. Así que ten tu editor abierto desde ahorita y acompaña cada sección con práctica real.

También te recomiendo que guardes todo lo que hagas en esta unidad en una carpeta especial, porque el código de canvas y SVG es de lo más reutilizable que existe. Ese logo en SVG que vas a dibujar hoy te puede servir mañana para tu portafolio. Esa función de JavaScript que dibuja una gráfica de barras la vas a querer copiar en tu primer proyecto de tablero de datos. Piensa en esta unidad como tu caja de herramientas visual: cada ejemplo es una herramienta que se queda contigo.

Para cerrar esta introducción, quiero que tengas clara la diferencia entre las tres estrellas de la unidad, porque a veces se confunden. Canvas es como una pizarra: dibujas con JavaScript, trazo por trazo, y lo que dibujas se queda como pintura, perfecto para gráficos que cambian todo el tiempo como juegos y animaciones. SVG es como dibujar con compás y regla sobre papel milimétrico: cada forma es un objeto matemático que puedes agrandar cuanto quieras sin que se deforme, perfecto para logos e iconos. Y las listas de definición son la forma elegante de emparejar términos con sus explicaciones, como un diccionario. Tres herramientas distintas, tres trabajos distintos, y al final de la unidad vas a saber exactamente cuándo usar cada una. Vamos a darle.

Explicación detallada

En esta sección vamos a desmenuzar cada tema de la unidad con calma y con ejemplos. No te preocupes si al principio canvas te parece raro o si SVG te suena a siglas de otro planeta: para el final de esta sección vas a manejar los dos con confianza. Empezamos con canvas, seguimos con SVG, luego vemos las listas avanzadas y de definición, y cerramos con los errores más comunes y las buenas prácticas que separan a un principiante de alguien que ya trabaja como profesional.

Canvas: tu pizarra para dibujar con código

Imagina que tienes una pizarra blanca colgada en la pared de tu página web. Está vacía, lista para que dibujes lo que quieras. Esa pizarra es la etiqueta canvas. Por sí sola no dibuja nada: es solo el espacio, el lienzo en blanco. El que toma el plumón y dibuja es JavaScript. Esta división de tareas es clave para entender canvas: HTML pone la pizarra, JavaScript hace los trazos.

La etiqueta se escribe así de simple:

```
<canvas id='lienzo' width='500' height='300'></canvas>
```

Fíjate en los tres atributos. El id es el nombre con el que JavaScript va a encontrar esa pizarra entre todo lo demás de la página, igual que le pones nombre a tus cosas para identificarlas. El width y el height son el ancho y el alto de la pizarra medidos en píxeles. Esto es importantísimo: si no pones width y height, el navegador le da un tamaño por defecto de 300 píxeles de ancho por 150 de alto, y además dibujar sin dimensiones claras causa problemas de escala que veremos en la sección de errores comunes. Acostúmbrate desde ya: todo canvas lleva sus dimensiones declaradas.

Un detalle que muchos pasan por alto: todo lo que escribas entre la etiqueta de apertura `<canvas>` y la de cierre `</canvas>` sirve como texto alternativo. Los navegadores viejos que no entienden canvas muestran ese texto en lugar de la pizarra. Es una cortesía mínima hacia esos casos y hacia la accesibilidad. Por ejemplo:

```
<canvas id='lienzo' width='500' height='300'>
  Tu navegador no soporta canvas. Actualízalo para ver este gráfico.
</canvas>
```

Hoy casi todos los navegadores soportan canvas, pero poner ese texto no cuesta nada y es una buena práctica.

Ahora, ¿cómo se dibuja? Aquí entra JavaScript. El proceso siempre tiene los mismos pasos, como una receta que vas a repetir hasta que te salga de memoria. Primero agarras la pizarra por su id. Después le pides su contexto de dibujo, que es como pedirle el juego de plumones. En canvas casi siempre trabajamos con el contexto 2d, que sirve para dibujar en dos dimensiones. Y luego das las órdenes de dibujo: rectángulos, líneas, círculos, texto.

Mira el ejemplo más simple posible, un rectángulo relleno:

```
<canvas id='lienzo' width='500' height='300'></canvas>
<script>
  var pizarra = document.getElementById('lienzo');
  var pincel = pizarra.getContext('2d');
  pincel.fillStyle = 'blue';
  pincel.fillRect(50, 50, 200, 100);
</script>
```

Vamos línea por línea porque cada una enseña algo. Con `getElementById` agarras el elemento canvas del HTML usando el id que le pusiste. Con `getContext('2d')` obtienes el objeto que tiene todos los métodos de dibujo; piensa en él como tu mano con el plumón. La propiedad `fillStyle` define el color de relleno que vas a usar de aquí en adelante, como elegir el plumón azul antes de dibujar. Y `fillRect` dibuja un rectángulo relleno con cuatro números: la coordenada x donde empieza, la coordenada y donde empieza, el ancho y el alto.

Las coordenadas en canvas merecen su propia explicación porque al principio confunden. El punto (0, 0), el origen, está en la esquina superior izquierda de la pizarra, no abajo como en las gráficas de matemáticas de la escuela. La x crece hacia la derecha y la y crece hacia abajo. Así que `fillRect(50, 50, 200, 100)` dibuja un rectángulo que empieza 50 píxeles a la derecha del borde izquierdo y 50 píxeles abajo del borde superior, con 200 de ancho y 100 de alto. Cuando dibujes tus primeras formas te recomiendo hacerte un dibujito en papel con estas coordenadas antes de escribir el código; te ahorra mucha confusión.

Los colores en canvas se escriben como en CSS: puedes usar nombres en inglés como 'red', 'blue' o 'green', códigos hexadecimales como '#ff0000', o valores rgb como 'rgb(255, 0, 0)'. Además del `fillStyle` para rellenos existe `strokeStyle` para el color de las líneas y los contornos, y `lineWidth` para el grosor de esas líneas. Con esas tres propiedades ya puedes dibujar formas con contorno de un color y relleno de otro, que es como se logran los dibujos que se ven profesionales.

Además de rectángulos, el contexto 2d sabe dibujar muchas cosas. Para líneas usas `beginPath` para empezar un trazo nuevo, `moveTo` para mover el plumón sin dibujar hasta un punto, `lineTo` para trazar hasta otro punto, y `stroke` para pintar la línea. Para círculos y arcos usas `arc`, que recibe el centro, el radio y los ángulos de inicio y fin. Para texto tienes `fillText` y `strokeText`, que escriben palabras en la pizarra en la posición que les digas, con la fuente que defines en la propiedad `font`. Con estas herramientas básicas ya puedes armar dibujos sorprendentemente elaborados: casas, caras, diagramas, gráficas de barras y mucho más.

Un ejemplo que junta varias técnicas: dibujar una carita feliz. Primero un círculo grande amarillo para la cara, luego dos círculos pequeños negros para los ojos, y un arco para la sonrisa. Cada parte es una orden sencilla; el truco está en combinarlas. Eso es lo bonito de canvas: los dibujos complejos no son más que muchas órdenes simples puestas en el orden correcto, como una receta de cocina donde cada paso es fácil pero el platillo final impresiona.

Ahora hablemos de por qué canvas es tan importante: es la base para juegos y gráficos dinámicos. Piensa en los juegos que corren en el navegador, esos donde un personaje se mueve y esquiva obstáculos. Por dentro, lo que pasa es que JavaScript borra la pizarra y la vuelve a dibujar muchas veces por segundo, cada vez con el personaje un poquito más adelante. Esa es la animación en canvas: dibujar, borrar, mover un poco, volver a dibujar, rapidísimo. También sirve para gráficas que se actualizan solas, como las de una página de finanzas donde los números cambian en vivo, o visualizaciones de datos donde el usuario mueve un control y la gráfica responde al instante.

Otra ventaja enorme de canvas es que dibuja con píxeles de forma directa, lo cual lo hace rapidísimo para escenas con miles de elementos, como partículas de un efecto de fuegos artificiales o los puntos de un mapa de calor. Cuando necesitas velocidad y movimiento constante, canvas es tu mejor amigo.

Pero ojo, canvas también tiene sus límites y es honesto conocerlos. Lo que dibujas en canvas se convierte en píxeles, como pintura sobre la pizarra: una vez pintado, el navegador ya no sabe que eso era un círculo o un rectángulo, solo ve puntitos de color. Eso significa que no puedes decirle después 'oye, mueve ese círculo a la derecha' porque para el navegador ya no hay círculo, hay píxeles. Si quieres mover algo, tienes que borrar todo y redibujar. También significa que el texto dibujado en canvas no lo pueden leer los lectores de pantalla ni seleccionar los usuarios, así que canvas no es el lugar para poner información importante solo como texto dibujado. Para eso están las etiquetas HTML normales.

En resumen de esta parte: canvas es tu pizarra, JavaScript es tu mano, y juntos te dan un espacio donde puedes dibujar cualquier cosa con código. Declara siempre sus dimensiones,

pon texto alternativo, aprende el sistema de coordenadas con la esquina superior izquierda como origen, y practica combinando formas simples para crear dibujos complejos. En la siguiente subsección conocerás a su primo elegante, SVG, y aprenderás cuándo conviene usar cada uno.

Estilo en canvas: más allá del color plano

Hasta ahora pintaste con colores planos, y está bien para empezar, pero canvas puede mucho más. Con unas pocas propiedades extra tus dibujos pasan de verse planos a verse profesionales: degradados, transparencias, sombras y líneas con estilo. Todo se controla con propiedades del contexto, igual que `fillStyle`.

Los degradados se crean con `createLinearGradient`, que define una transición de color entre dos puntos. Le pasas las coordenadas de inicio y fin, luego agregas los colores con `addColorStop` indicando en qué punto del recorrido va cada uno, del 0 al 1. El resultado lo asignas a `fillStyle` como si fuera un color normal:

```
var degradado = pincel.createLinearGradient(0, 0, 300, 0);
degradado.addColorStop(0, 'navy');
degradado.addColorStop(1, 'skyblue');
pincel.fillStyle = degradado;
pincel.fillRect(50, 50, 300, 100);
```

Esto pinta un rectángulo que va del azul marino al azul cielo de izquierda a derecha. Los degradados sirven para fondos, botones con volumen y cielos en paisajes. También existe `createRadialGradient` para degradados circulares, perfectos para soles, focos de luz o esferas con volumen.

La transparencia se controla con `globalAlpha`, un número entre 0 y 1 que define qué tan opaco es todo lo que dibujes después. Con `globalAlpha` en 0.5 dibujas a la mitad de opacidad, ideal para superponer formas y crear efectos de cristal o sombras suaves. Solo recuerda regresarla a 1 cuando termines, porque afecta a todo lo que sigue y es fácil olvidarlo y luego no entender por qué tus colores se ven apagados.

Las sombras le dan profundidad a tus formas con tres propiedades: `shadowColor` define el color de la sombra, `shadowBlur` qué tan difuminada está y `shadowOffsetX` con `shadowOffsetY` hacia dónde se proyecta. Con una sombra sutil debajo de un rectángulo logras el efecto de tarjeta flotante que se ve en tantas interfaces modernas.

Para las líneas tienes `lineWidth` para el grosor, que ya conoces, y además `lineCap` que define cómo se ven los extremos: `butt` los deja cortados en seco, `round` los redondea y `square` los extiende un poco. El redondeado es el favorito para dibujar a mano alzada o gráficas con estilo suave. Y para el texto, la propiedad `font` acepta los mismos valores que en CSS, así que puedes usar negritas con `'bold 24px Arial'` o cursivas con `'italic 18px Georgia'`.

Mira un ejemplo que combina varias de estas técnicas en una tarjeta de presentación dibujada en canvas:


```

<canvas id='tarjeta' width='400' height='220'>
  Tarjeta de presentación dibujada en canvas.
</canvas>
<script>
  var pizarra = document.getElementById('tarjeta');
  var pincel = pizarra.getContext('2d');

  // Fondo con degradado horizontal
  var fondo = pincel.createLinearGradient(0, 0, 400, 0);
  fondo.addColorStop(0, '#0f2a5c');
  fondo.addColorStop(1, '#2a6fd6');
  pincel.fillStyle = fondo;
  pincel.fillRect(0, 0, 400, 220);

  // Tarjeta blanca con sombra flotante
  pincel.shadowColor = 'rgba(0, 0, 0, 0.4)';
  pincel.shadowBlur = 20;
  pincel.shadowOffsetY = 8;
  pincel.fillStyle = 'white';
  pincel.fillRect(40, 40, 320, 140);

  // Quitar la sombra para lo que sigue
  pincel.shadowColor = 'transparent';
  pincel.shadowBlur = 0;
  pincel.shadowOffsetY = 0;

  // Círculo decorativo semitransparente
  pincel.globalAlpha = 0.25;
  pincel.fillStyle = '#2a6fd6';
  pincel.beginPath();
  pincel.arc(320, 60, 45, 0, Math.PI * 2);
  pincel.fill();
  pincel.globalAlpha = 1;

  // Textos de la tarjeta
  pincel.fillStyle = '#0f2a5c';
  pincel.font = 'bold 26px Arial';
  pincel.fillText('Lucas EGA Academy', 70, 100);
  pincel.fillStyle = '#555555';
  pincel.font = '16px Arial';
  pincel.fillText('Cursos de tecnología en español', 70, 130);
  pincel.fillText('lucasegaacademy.com', 70, 155);
</script>

```

Observa el orden de las operaciones: primero el fondo, luego la tarjeta con sombra, después se apaga la sombra para que no afecte lo demás, luego el círculo decorativo con transparencia y al final los textos. Cada técnica de estilo se activa, se usa y se desactiva de forma consciente. Este control fino es lo que separa un dibujo amateur de uno profesional, y ya lo tienes en tus manos.

SVG: gráficos que nunca se pixelan

Si canvas es una pizarra donde pintas con píxeles, SVG es dibujar con compás, regla y fórmulas matemáticas sobre un papel que puedes agrandar al tamaño de un edificio sin que se deforme nada. SVG significa Scalable Vector Graphics, o sea gráficos vectoriales escalables. La palabra clave es vectorial: en lugar de guardar puntitos de color, SVG guarda instrucciones matemáticas del tipo 'dibuja un círculo de radio 40 en esta posición con este color'. Y como son matemáticas, puedes agrandar el dibujo cuanto quieras y siempre se ve nítido, perfecto, sin esos cuadritos borrosos que aparecen cuando agrandas una foto demasiado.

Para entender la diferencia, piensa en dos formas de describir una casa a un amigo. La forma de mapa de bits, que es como funcionan las fotos JPG y PNG, sería decirle el color de cada cuadrito de una cuadrícula gigante: 'el cuadrito 1 es blanco, el cuadrito 2 es blanco, el cuadrito 3 es café...'. Si luego quieres la casa al doble de tamaño, tienes que inventarte cuadritos intermedios y se ve borroso. La forma vectorial sería decirle 'dibuja un cuadrado de este tamaño para la base y un triángulo encima para el techo'. Con esa descripción tu amigo puede dibujar la casa en una servilleta o en una barda y siempre se ve bien. Eso es exactamente lo que hace SVG: describe formas, no píxeles.

La etiqueta base es <svg> y dentro de ella pones las formas. Cada forma es una etiqueta con sus atributos. Las más usadas son circle para círculos, rect para rectángulos, line para líneas, ellipse para óvalos, polygon para figuras de varios lados y path para trazos libres y complejos. Mira un ejemplo sencillo, un círculo:

```
<svg width='200' height='200'>
  <circle cx='100' cy='100' r='80' fill='red' />
</svg>
```

El svg define un espacio de 200 por 200 píxeles. Dentro, circle dibuja un círculo cuyo centro está en las coordenadas cx='100' y cy='100', con radio r='80', relleno de color rojo con fill='red'. Fíjate que estas etiquetas se cierran solas con /> porque no llevan contenido adentro, igual que la etiqueta img que ya conoces.

Cada forma tiene sus atributos propios pero comparten los más importantes. `fill` es el color de relleno, `stroke` es el color del contorno y `stroke-width` es el grosor de ese contorno. Si quieres una forma solo con contorno y sin relleno, pones `fill='none'`. Con estas tres propiedades controlas la apariencia de todo lo que dibujes en SVG, igual que en canvas pero declarado directamente en el HTML, sin JavaScript.

Veamos las formas una por una con ejemplos mentales que puedes probar. El `rect` lleva `x` e `y` para su esquina superior izquierda, `width` y `height` para su tamaño, y opcionalmente `rx` para redondear las esquinas, que es como se hacen los botones bonitos. El `line` lleva `x1`, `y1` para el punto inicial y `x2`, `y2` para el punto final. El `ellipse` es como el círculo pero con dos radios, `rx` horizontal y `ry` vertical, perfecto para óvalos. El `polygon` recibe una lista de puntos en el atributo `points`, como `'100,10 40,190 160,190'` para un triángulo, y cierra la figura solo. Y el `path` es el más poderoso: con comandos de una letra como `M` para moverse, `L` para trazar líneas y `Z` para cerrar, puedes dibujar cualquier silueta, desde una estrella hasta el contorno de un país.

El `path` merece un ejemplo porque al principio impone respeto pero es más sencillo de lo que parece:

```
<svg width='200' height='200'>
  <path d='M 20 180 L 100 20 L 180 180 Z' fill='none' stroke='blue'
stroke-width='4' />
</svg>
```

El atributo `d` contiene las instrucciones: `M 20 180` mueve el lápiz al punto (20, 180) sin dibujar, `L 100 20` traza una línea hasta (100, 20), `L 180 180` traza otra hasta (180, 180), y `Z` cierra la figura volviendo al inicio. El resultado es un triángulo con contorno azul. Una vez que entiendes que el `path` es solo una lista de órdenes encadenadas, deja de dar miedo.

Ahora, un concepto de SVG que causa mucha confusión al principio: el `viewBox`. El `viewBox` define el sistema de coordenadas interno del dibujo, independiente del tamaño en píxeles que ocupa en la página. Se escribe como `viewBox='0 0 200 200'`, que significa 'mi dibujo vive en un mundo de 200 por 200 unidades que empieza en (0,0)'. La magia es que si luego pones `width='400' height='400'`, el dibujo se escala al doble automáticamente y se sigue viendo perfecto. El `viewBox` es lo que hace que SVG sea verdaderamente escalable: separas el diseño del tamaño final. Si omites el `viewBox`, el SVG usa píxeles directos y pierdes parte de esa flexibilidad, así que acostúmbrate a incluirlo siempre.

¿Por qué no se pixelan los SVG? Porque el navegador no guarda una imagen fija: cada vez que necesita mostrar el SVG, recalcula las formas matemáticas al tamaño necesario. En una

pantalla de celular dibuja el círculo con los píxeles justos para ese tamaño; en un monitor 4K lo recalcula con muchísimos más píxeles. Siempre nítido, siempre perfecto. Por eso los logos profesionales de las empresas grandes están en SVG: el mismo archivo se ve impecable en el favicon chiquito de la pestaña del navegador y en el espectacular de la autopista.

Esto nos lleva a la pregunta del millón: ¿cuándo uso SVG, cuándo canvas y cuándo una imagen normal con img? La respuesta depende de lo que necesites. Usa SVG cuando el gráfico está hecho de formas claras y necesita verse nítido a cualquier tamaño: logos, iconos, diagramas, ilustraciones simples. Además, como cada forma de SVG es una etiqueta del HTML, puedes darle estilos con CSS y hasta animarlas o hacerlas responder a clics, cosa que con canvas es mucho más trabajosa. Usa canvas cuando el gráfico cambia constantemente o tiene miles de elementos: juegos, animaciones complejas, gráficas en tiempo real, efectos de partículas. Y usa img con JPG o PNG cuando es una fotografía o una imagen muy detallada con sombras y texturas reales, porque ni SVG ni canvas están hechos para fotos.

Un truco profesional: los iconos de tu sitio deberían ser SVG casi siempre. Antes se usaban imágenes PNG para cada icono, y en pantallas modernas se veían borrosos o había que hacer varias versiones de cada tamaño. Con SVG haces el icono una vez y se ve perfecto en todos lados, pesa poquísimo y hasta le puedes cambiar el color con CSS cuando el usuario pasa el mouse encima. Cuando veas una página bien hecha con iconos nítidos, casi seguro están usando SVG por dentro.

Para cerrar esta subsección, piensa en SVG como tu juego de geometría digital: compás para los círculos, regla para las líneas, y el path como tu lápiz de dibujo libre. Todo lo que dibujes son matemáticas, así que nada se pixela nunca. Declara tu viewBox, usa fill y stroke para darle vida a las formas, y reserva SVG para logos, iconos y diagramas mientras dejas los juegos y las animaciones intensas para canvas. Con estas dos herramientas ya cubres casi cualquier necesidad gráfica de una página web moderna.

Listas avanzadas y de definición: orden para tus ideas

Después de tanto dibujo, cambiamos de tercio hacia algo igual de útil pero menos vistoso: las listas especializadas. Ya conoces las listas básicas, ul para listas sin orden y ol para listas numeradas, con sus elementos li. Ahora vamos a ver su lado avanzado: las listas de definición con dl, dt y dd, las listas anidadas complejas y los atributos start y type de las listas ordenadas. Puede sonar menos emocionante que dibujar en canvas, pero te aseguro que una página con listas bien usadas se lee diez veces mejor que una con párrafos interminables.

Empecemos con la estrella de la clase 21: la lista de definición. Su estructura usa tres etiquetas que trabajan en equipo. La etiqueta `dl` es el contenedor, la lista de definición completa, como la pasta de un diccionario. Dentro de ella, `dt` marca cada término, la palabra que se va a definir, como la entrada del diccionario en negritas. Y `dd` contiene la definición, la explicación del término, como el texto que viene después de la palabra en el diccionario. Mira qué elegante se ve:

```
<dl>
  <dt>HTML</dt>
  <dd>Lenguaje de marcado para estructurar páginas web.</dd>
  <dt>CSS</dt>
  <dd>Lenguaje para dar estilo y diseño a las páginas web.</dd>
</dl>
```

El navegador muestra cada `dt` destacado y cada `dd` con una sangría, como un diccionario de verdad. La analogía del diccionario es perfecta para recordar el uso: cada vez que tengas pares de término y explicación, piensa en `dl`. Y hay una regla importante: los `dt` y `dd` siempre van dentro de un `dl`, nunca sueltos, y cada `dd` pertenece al `dt` que tiene justo antes. Incluso puedes poner varios `dt` seguidos si comparten la misma definición, o varios `dd` para un solo `dt` si un término tiene varias acepciones, igual que en un diccionario real donde una palabra puede tener varios significados.

¿Para qué sirve en la vida real? Para glosarios, que ya vimos, pero también para preguntas frecuentes, las famosas FAQ. Cada pregunta es un `dt` y cada respuesta es un `dd`. También sirven para fichas de datos, como la ficha de un producto donde el término es 'Precio' y la definición es '\$299', o el currículum donde el término es 'Experiencia' y la definición describe tus trabajos. Cada vez que veas información en pares de etiqueta y valor, `dl` es la etiqueta semánticamente correcta, mucho mejor que forzar una tabla o una lista normal para algo que no son.

Ahora las listas anidadas complejas. Anidar significa meter una lista dentro de un elemento `li` de otra lista, creando niveles como las ramas de un árbol. Esto sirve para menús de navegación con submenús, índices de contenido con secciones y subsecciones, y organigramas. La regla de oro es sencilla pero se rompe mucho: la lista anidada siempre va dentro del `li`, no después de él ni como hermana. Mira la estructura correcta:

```
<ul>
  <li>Inicio</li>
  <li>Cursos
    <ul>
      <li>HTML5</li>
      <li>CSS3</li>
      <li>JavaScript</li>
    </ul>
  </li>
  <li>Contacto</li>
</ul>
```

Fíjate que el segundo ul está dentro del li de 'Cursos', antes de su etiqueta de cierre. Así el navegador entiende que esos tres cursos son hijos de 'Cursos'. Si pusieras el ul fuera del li, la estructura se rompe y el significado se pierde, aunque visualmente a veces se vea parecido. Los lectores de pantalla y los buscadores sí notan la diferencia, así que hazlo bien desde el principio. Puedes anidar tantos niveles como quieras y hasta mezclar ul dentro de ol y viceversa, aunque más de tres niveles se vuelve difícil de leer y conviene simplificar el diseño.

Pasemos a los atributos start y type de las listas ordenadas. El atributo type cambia el estilo de la numeración: con type='1' tienes números normales, con type='A' letras mayúsculas, con type='a' letras minúsculas, con type='I' números romanos mayúsculos y con type='i' romanos minúsculos. Esto es ideal para documentos formales, exámenes o índices donde las secciones usan letras y las subsecciones números. El atributo start indica desde qué número o letra empieza la cuenta. Por ejemplo, <ol start='5'> empieza en el 5, y <ol type='A' start='3'> empieza en la letra C. Esto sirve cuando una lista continúa algo que empezó antes, como los pasos de un tutorial divididos en varias páginas donde la segunda página retoma la numeración.

Un detalle moderno: en HTML5 se prefiere controlar la apariencia de las listas con CSS usando list-style-type, pero los atributos type y start siguen siendo válidos y muy prácticos para casos simples. Conócelos y úsalos cuando te ahorren trabajo, y cuando quieras diseños más elaborados pásate a CSS sin culpa.

Las listas de definición en acción: preguntas frecuentes

Ya viste que dl sirve para glosarios, pero su uso más frecuente en sitios reales son las preguntas frecuentes, las famosas FAQ que aparecen en casi todas las páginas de servicios. La estructura encaja perfecto: cada pregunta es un dt y cada respuesta es un dd. Usar dl en lugar de una bola de divs o párrafos tiene ventajas concretas: los lectores de pantalla anuncian

la relación pregunta-respuesta, los buscadores entienden que es contenido de preguntas y respuestas, y tu código refleja lo que el contenido realmente es.

Mira cómo se arma una sección de preguntas frecuentes bien estructurada:

```
<h2>Preguntas frecuentes</h2>
<dl>
  <dt>¿Necesito experiencia previa para tomar el curso?</dt>
  <dd>No. El curso empieza desde cero y cada unidad construye
    sobre la anterior. Si sabes usar un navegador, puedes empezar.</dd>

  <dt>¿Cuánto tiempo me toma terminarlo?</dt>
  <dd>Depende de tu ritmo, pero dedicándole una hora diaria
    terminas en unas seis semanas.</dd>

  <dt>¿Recibo un certificado al finalizar?</dt>
  <dd>Sí. Al completar todas las unidades y el proyecto final
    recibes tu certificado digital de Lucas EGA Academy.</dd>

  <dt>¿Puedo ver las clases en mi celular?</dt>
  <dd>Claro. Todo el contenido está diseñado para verse bien
    en celular, tableta y computadora.</dd>
</dl>
```

Fíjate que las preguntas usan los signos de apertura ¿que el español exige, y cada respuesta es directa y completa. Cuando más adelante aprendas CSS, podrás darle a los dt un estilo de pregunta destacada y a los dd un estilo de respuesta, e incluso hacer que las respuestas se muestren y se oculten al hacer clic en la pregunta. Pero la base semántica, la que hace que todo funcione bien por dentro, ya está lista desde ahorita con dl.

Otro uso muy común son las fichas de datos, como la ficha técnica de un producto o los datos de contacto de una empresa. Cada etiqueta del dato es un dt y cada valor es un dd:

```
<h2>Ficha del curso</h2>
<dl>
  <dt>Duración</dt>
  <dd>6 semanas</dd>
  <dt>Nivel</dt>
  <dd>Principiante</dd>
  <dt>Modalidad</dt>
  <dd>En línea, a tu ritmo</dd>
  <dt>Precio</dt>
  <dd>Gratis</dd>
</dl>
```

Antes de que existiera esta estructura, la gente armaba estas fichas con tablas forzadas o con párrafos llenos de negritas, y el resultado era código confuso. Con dl el significado queda clarísimo: esto es una lista de pares etiqueta-valor. Cada vez que en un diseño veas dos columnas donde la izquierda nombra algo y la derecha lo describe, piensa en dl antes que en cualquier otra etiqueta.

Errores comunes que debes evitar

En canvas, el error número uno es no declarar width y height y luego intentar controlar el tamaño con CSS. Si pones un canvas sin dimensiones y luego le das tamaño con CSS, el dibujo se estira como chicle y se ve deformado, porque el CSS solo agranda la pizarra pero el área de dibujo interna sigue siendo la de por defecto. La regla es clara: las dimensiones del canvas se declaran con los atributos width y height en la etiqueta, y el CSS solo se usa para casos especiales sabiendo lo que haces.

El segundo error típico de canvas es olvidar el beginPath antes de dibujar una nueva forma con trazos. Sin beginPath, los trazos se acumulan y de pronto tus líneas se conectan de formas rarísimas que no planeaste. Cada vez que empieces una figura nueva con moveTo y lineTo, abre con beginPath. Es como levantar el plumón antes de empezar otro dibujo.

El tercer error es dibujar texto importante solo dentro del canvas. Recuerda que ese texto se vuelve píxeles: no se puede seleccionar, no lo leen los lectores de pantalla y los buscadores lo ignoran. Si el texto es información que el usuario necesita leer o copiar, ponlo en HTML normal y usa canvas solo para lo visual.

En SVG, el error más común es olvidar el viewBox y luego sorprenderse de que el dibujo no escala bien o se recorta raro. Sin viewBox el SVG trabaja en píxeles fijos y pierdes la magia de la escalabilidad. Inclúyelo siempre, aunque al principio te parezca un adorno misterioso.

Otro error frecuente en SVG es usarlo para fotografías o imágenes con miles de detalles. Un retrato convertido a SVG con miles de paths pesa una barbaridad y se dibuja lento. SVG es para formas claras y simples; las fotos son territorio de JPG y PNG con la etiqueta `img`.

En las listas, el error clásico es anidar mal: poner la lista hija fuera del `li` en lugar de dentro. Visualmente a veces ni se nota, pero la estructura queda rota para las tecnologías de asistencia. El segundo error es usar listas para cosas que no son listas, como meter párrafos sueltos en `li` solo para aprovechar las viñetas. Y el tercero es olvidar que `dt` y `dd` solo viven dentro de `dl`; ponerlos sueltos es código inválido.

Buenas prácticas de un profesional

Primera: elige la herramienta correcta para cada trabajo. ¿Logo o icono que debe verse nítido siempre? SVG. ¿Juego, animación o gráfica que cambia en vivo? Canvas. ¿Foto o imagen detallada? `img` con JPG o PNG. ¿Pares de término y definición? `dl`. Esta decisión es la mitad del trabajo bien hecho.

Segunda: en canvas, organiza tu código de dibujo en funciones con nombres claros, como `dibujarCara` o `dibujarGrafica`. Cuando tu dibujo crezca, agradecerás poder leer funciones en lugar de cien líneas seguidas de órdenes sueltas. Además esas funciones las reutilizas en otros proyectos.

Tercera: siempre pon texto alternativo dentro de la etiqueta canvas y usa `title` o descripciones en SVG cuando el gráfico transmite información. La accesibilidad no es un lujo, es parte del trabajo bien hecho.

Cuarta: mantén tus SVG limpios y ordenados. Si los creas en un programa de diseño y los pegas en tu HTML, revisa que no traigan basura innecesaria. Un SVG limpio pesa poco y carga rápido.

Quinta: para las listas, piensa en la estructura antes que en la apariencia. Primero arma la jerarquía correcta de la información con las etiquetas adecuadas, y después preocúpate por cómo se ve con CSS. Una lista bien estructurada se ve bien con cualquier estilo; una lista mal estructurada no la salva ningún diseño.

Canvas más a fondo: tu primera animación

Ya sabes dibujar formas estáticas en canvas. Ahora viene la parte emocionante: hacer que se muevan. La animación en canvas se basa en una idea muy simple que ya mencionamos: dibujar, borrar, mover un poquito y volver a dibujar, muchas veces por segundo. Es

exactamente como las caricaturas de antes, que eran miles de dibujos casi idénticos pasados rapidísimo para crear la ilusión de movimiento.

Para borrar la pizarra existe el método `clearRect`, que limpia un rectángulo del canvas dejándolo transparente. Normalmente lo usas para limpiar todo el canvas al inicio de cada cuadro de la animación: `pincel.clearRect(0, 0, 500, 300)` borra una pizarra de 500 por 300. Después de borrar, dibujas todo de nuevo pero con las posiciones ligeramente cambiadas. Si repites esto unas 60 veces por segundo, el ojo humano ve movimiento fluido.

JavaScript trae una función perfecta para esto: `requestAnimationFrame`. Le pasas una función con tus instrucciones de dibujo y el navegador la ejecuta justo antes de pintar el siguiente cuadro en pantalla, unas 60 veces por segundo, y de forma eficiente porque se pausa sola cuando la pestaña no está visible. El patrón clásico es que la función se llame a sí misma al final, creando un ciclo infinito de dibujo. Mira un ejemplo completo: una pelota que rebota de lado a lado.

```

<canvas id='rebote' width='500' height='200'>
  Animación de una pelota rebotando.
</canvas>
<script>
  var pizarra = document.getElementById('rebote');
  var pincel = pizarra.getContext('2d');

  // Estado de la pelota: posición y velocidad
  var x = 50;           // empieza a la izquierda
  var velocidad = 3;    // píxeles que avanza por cuadro

  function dibujarCuadro() {
    // Paso 1: borrar todo lo del cuadro anterior
    pincel.clearRect(0, 0, 500, 200);

    // Paso 2: dibujar la pelota en su posición actual
    pincel.fillStyle = 'crimson';
    pincel.beginPath();
    pincel.arc(x, 100, 30, 0, Math.PI * 2);
    pincel.fill();

    // Paso 3: mover la pelota para el siguiente cuadro
    x = x + velocidad;

    // Paso 4: si toca un borde, invertir la dirección
    if (x > 470 || x < 30) {
      velocidad = -velocidad;
    }

    // Paso 5: pedir el siguiente cuadro, el ciclo continúa
    requestAnimationFrame(dibujarCuadro);
  }

  // Arrancar la animación
  dibujarCuadro();
</script>

```

Analiza lo que pasa: la variable `x` guarda la posición y vive fuera de la función para que no se reinicie. Cada cuadro borra, dibuja, mueve y revisa los bordes. Cuando la pelota llega al borde derecho (`x` mayor a 470, que es 500 menos el radio 30) o al izquierdo (`x` menor a 30), la velocidad cambia de signo y la pelota regresa. Este patrón de estado más ciclo de dibujo es la base de todos los juegos 2d: el personaje, los enemigos y los obstáculos no son más que variables que cambian y se redibujan.

Un consejo de organización: cuando tus dibujos crezcan, separa el código en funciones como `dibujarFondo`, `dibujarPelota` y `actualizarPosicion`. Así tu ciclo principal queda limpio y legible, y cada parte se puede probar por separado. Los programadores profesionales de juegos organizan su código exactamente así, aunque sus juegos tengan miles de líneas.

SVG más a fondo: grupos, texto y estilos con CSS

En la subsección de SVG viste las formas básicas. Ahora vamos a ver tres herramientas que lo vuelven mucho más poderoso: los grupos, el texto integrado y la capacidad de darle estilo con CSS como a cualquier otro elemento HTML.

La etiqueta `g` agrupa varias formas para tratarlas como una sola unidad. Esto sirve para mover, escalar o dar estilo a un conjunto completo sin repetir atributos en cada forma. Con el atributo `transform` puedes trasladar todo el grupo: `transform='translate(50, 30)'` mueve el grupo 50 píxeles a la derecha y 30 hacia abajo. Imagina que dibujaste una casita con cinco formas y quieres poner tres casitas en distintos lugares: las agrupas una vez y luego las reutilizas con distintas traslaciones en lugar de recalcular cada coordenada a mano.

El texto en SVG merece atención porque, a diferencia del texto dibujado en canvas, el texto de SVG sigue siendo texto de verdad: se puede seleccionar, lo leen los lectores de pantalla y los buscadores lo indexan. La etiqueta `text` coloca palabras en las coordenadas que le digas, y con `text-anchor` las alineas: `start` a la izquierda, `middle` al centro y `end` a la derecha del punto. Con `font-size`, `font-family` y `fill` controlas su apariencia, igual que en CSS.

Y hablando de CSS: como el SVG escrito directamente en el HTML es parte del documento, puedes darle estilos con CSS igual que a un `div` o un párrafo. Puedes cambiar el `fill` de un icono cuando el usuario pasa el mouse encima con la pseudoclase `hover`, o animar formas con transiciones. Esto es algo que canvas no puede hacer tan fácil, y es una razón de peso para preferir SVG en iconos e ilustraciones interactivas. Mira un ejemplo que junta grupos, texto y estilo:

```

<svg width='240' height='120' viewBox='0 0 240 120'>
  <!-- Grupo de la insignia: círculo y estrella juntos -->
  <g transform='translate(60, 60)'>
    <circle r='45' fill='gold' stroke='darkorange' stroke-width='4' />
    <polygon points='0,-28 8,-9 28,-9 12,4 18,24 0,12 -18,24 -12,4 -28,-9 -8,-9'
      fill='darkorange' />
  </g>
  <!-- Texto al lado de la insignia -->
  <text x='125' y='55' font-size='20' font-family='Arial' fill='#333333'>
    Nivel
  </text>
  <text x='125' y='85' font-size='32' font-family='Arial' fill='#1e5aa8'>
    Experto
  </text>
</svg>

```

El grupo g contiene el círculo y la estrella de la insignia, y con un solo translate los colocas donde quieras. Si mañana decides mover la insignia, cambias dos números en lugar de recalcular todas las coordenadas. Esa es la magia de pensar en grupos: diseñas piezas y las acomodas como bloques.

Guía de decisión: canvas, SVG o imagen

Con tres herramientas gráficas en tu caja, la pregunta ya no es cómo dibujar sino qué usar en cada caso. Aquí tienes una guía práctica con los casos más comunes que te vas a encontrar en proyectos reales.

- Logo de una empresa en el encabezado del sitio: SVG. Se ve nítido en celular y en monitor grande, pesa poco y puedes cambiarle el color con CSS para el modo oscuro.
- Iconos de redes sociales en el pie de página: SVG. Un solo archivo por icono sirve para todos los tamaños y se ven perfectos en pantallas de alta resolución.
- Fotografía de un producto en una tienda en línea: img con JPG. Las fotos tienen miles de tonos y texturas que ni SVG ni canvas manejan bien.
- Juego de naves donde disparas a enemigos que se mueven: canvas. Necesitas redibujar decenas de objetos 60 veces por segundo y canvas está optimizado justo para eso.
- Gráfica de pastel estática en un reporte anual: SVG. Son pocas formas claras, se ve nítida al imprimir y el texto de las etiquetas sigue siendo seleccionable.

- Gráfica de temperatura que se actualiza cada segundo con datos de sensores: canvas. El redibujado constante es su territorio natural.
- Icono de menú hamburguesa que se anima al abrirse: SVG. Tres líneas que con CSS se transforman en una X, todo ligero y nítido.
- Diagrama de flujo de un proceso explicado en un artículo: SVG. Formas claras, texto legible y escalable para quien lo quiera ampliar.
- Efecto de confeti que explota al ganar un premio en la página: canvas. Cientos de partículas moviéndose a la vez piden la velocidad de los píxeles directos.
- Ilustración decorativa compleja hecha por un diseñador: depende. Si son formas vectoriales simples, SVG; si es una ilustración con texturas fotorrealistas, mejor PNG con img.

La regla general cabe en una frase: si el gráfico está quieto o cambia poco y está hecho de formas claras, SVG; si se mueve todo el tiempo o tiene miles de elementos, canvas; si es una foto o imagen con detalle realista, img. Con esta guía ya decides como profesional y dejas de adivinar.

Ejemplos paso a paso

La teoría ya quedó clara, ahora toca ver código de verdad. Cada ejemplo viene completo y comentado para que lo copies a tu editor, lo abras en el navegador y juegues con él. Te recomiendo que después de hacerlo funcionar cambies valores a propósito: colores, tamaños, posiciones. Esa experimentación es donde ocurre el verdadero aprendizaje.

Ejemplo 1: página con canvas que dibuja formas básicas con JavaScript

Este ejemplo crea una pizarra de 500 por 300 píxeles y dibuja en ella un cielo con sol, pasto y una casa usando solo formas básicas. Cada parte del dibujo está comentada para que veas qué hace cada orden.

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Mi primer dibujo en canvas</title>
</head>
<body>
  <h1>Paisaje dibujado con código</h1>
```

```

<!-- La pizarra: siempre con id y dimensiones declaradas -->
<!-- El texto interior es alternativo por si el navegador no soporta canvas -->
<canvas id='paisaje' width='500' height='300'>
  Tu navegador no soporta canvas. Prueba con uno más moderno.
</canvas>

<script>
  // Paso 1: agarrar la pizarra por su id
  var pizarra = document.getElementById('paisaje');

  // Paso 2: pedir el contexto de dibujo en 2 dimensiones
  var pincel = pizarra.getContext('2d');

  // Paso 3: pintar el cielo, un rectángulo azul que cubre todo
  pincel.fillStyle = 'skyblue';
  pincel.fillRect(0, 0, 500, 300);

  // Paso 4: dibujar el sol, un círculo amarillo en la esquina
  pincel.fillStyle = 'yellow';
  pincel.beginPath();
  pincel.arc(420, 60, 40, 0, Math.PI * 2);
  pincel.fill();

  // Paso 5: pintar el pasto, un rectángulo verde abajo
  pincel.fillStyle = 'green';
  pincel.fillRect(0, 220, 500, 80);

  // Paso 6: la base de la casa, un rectángulo café
  pincel.fillStyle = 'brown';
  pincel.fillRect(180, 140, 140, 80);

  // Paso 7: el techo, un triángulo dibujado con trazos
  pincel.fillStyle = 'red';
  pincel.beginPath();
  pincel.moveTo(170, 140); // esquina izquierda del techo
  pincel.lineTo(250, 80); // punta del techo
  pincel.lineTo(330, 140); // esquina derecha del techo
  pincel.closePath();
  pincel.fill();

  // Paso 8: la puerta, un rectángulo oscuro centrado
  pincel.fillStyle = '#333333';
  pincel.fillRect(235, 175, 30, 45);

```

```
// Paso 9: escribir un texto sobre el dibujo
pincel.fillStyle = 'white';
pincel.font = '20px Arial';
pincel.fillText('Mi casa', 20, 40);
</script>
</body>
</html>
```

Fíjate en el orden: primero el fondo y luego los objetos de atrás hacia adelante, igual que pintarías un cuadro real. El sol se dibuja con `arc` usando `Math.PI * 2` para completar el círculo entero. El techo usa `beginPath` con `moveTo` y `lineTo`, y `closePath` cierra la figura antes de rellenarla. El texto se escribe al final con `fillText` indicando la posición. Si cambias los números de `fillRect` verás cómo se mueve cada pieza: esa es la mejor forma de entender las coordenadas.

Ejemplo 2: logo en SVG con varias formas

Ahora un logo ficticio para una academia, construido solo con formas SVG. Incluye `viewBox` para que escale perfecto, un círculo de fondo, un libro simplificado con `rect` y `polygon`, y texto. Todo declarado en el HTML, sin JavaScript.


```

<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Logo en SVG</title>
</head>
<body>
  <h1>Logo de mi academia</h1>

  <!-- viewBox define el mundo interno de 200x200 unidades -->
  <!-- width y height definen el tamaño en la página: escala sin pixelarse -->
  <svg width='300' height='300' viewBox='0 0 200 200'>
    <!-- Fondo: círculo grande azul -->
    <circle cx='100' cy='100' r='95' fill='#1e5aa8' />

    <!-- Anillo decorativo: círculo solo con contorno blanco -->
    <circle cx='100' cy='100' r='80' fill='none'
      stroke='white' stroke-width='3' />

    <!-- Libro abierto: polígono con 4 puntos -->
    <polygon points='60,80 100,95 140,80 140,120 100,135 60,120'
      fill='white' />

    <!-- Lomo del libro: línea vertical al centro -->
    <line x1='100' y1='95' x2='100' y2='135'
      stroke='#1e5aa8' stroke-width='3' />

    <!-- Texto del logo debajo del libro -->
    <text x='100' y='170' text-anchor='middle'
      fill='white' font-size='18' font-family='Arial'>
      MI ACADEMIA
    </text>
  </svg>

  <p>
    Este mismo logo se ve nítido en 30 píxeles o en 3000.
    Prueba cambiar width y height y compruébalo.
  </p>
</body>
</html>

```

Observa cómo cada forma es una etiqueta legible: no necesitas adivinar qué hace cada cosa. El text-anchor='middle' centra el texto en la coordenada x que le das. El atributo points del

polygon lista los vértices en orden. Y lo más importante: cambia width='300' height='300' por otros valores y verás que el logo nunca se deforma ni se pixela, gracias al viewBox.

Ejemplo 3: glosario de términos con dl

Un glosario de términos web usando lista de definición. Cada término es un dt y su explicación un dd. Incluye un caso con dos dt que comparten definición y un término con dos dd, para que veas la flexibilidad.

```

<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Glosario web</title>
</head>
<body>
  <h1>Glosario de términos web</h1>

  <dl>
    <!-- Par simple: un término, una definición -->
    <dt>HTML</dt>
    <dd>Lenguaje de marcado para estructurar el contenido
      de las páginas web.</dd>

    <dt>CSS</dt>
    <dd>Lenguaje para definir la presentación visual:
      colores, tamaños y posiciones.</dd>

    <!-- Dos términos que comparten la misma definición -->
    <dt>JS</dt>
    <dt>JavaScript</dt>
    <dd>Lenguaje de programación que le da interactividad
      a las páginas web.</dd>

    <!-- Un término con dos definiciones o acepciones -->
    <dt>Responsive</dt>
    <dd>Diseño que se adapta a distintos tamaños de pantalla.</dd>
    <dd>Enfoque que prioriza la experiencia en celulares
      sin descuidar la computadora.</dd>

    <dt>SVG</dt>
    <dd>Formato de gráficos vectoriales que se dibujan con
      código y nunca se pixelan.</dd>
  </dl>
</body>
</html>

```

El navegador presenta cada dt resaltado y cada dd con sangría, logrando el aspecto de diccionario sin ningún CSS. Esta estructura además ayuda a los buscadores a entender que se trata de definiciones, lo cual es un plus para el posicionamiento de páginas educativas.

Ejemplo 4: menú multinivel con listas anidadas

Un menú de navegación de tres niveles para el sitio de una academia. La lista anidada va siempre dentro del li padre, y se mezclan ul para que veas que la estructura se mantiene clara.

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Menú multinivel</title>
</head>
<body>
  <nav>
    <h2>Mapa del sitio</h2>
    <ul>
      <li>Inicio</li>
      <li>Cursos
        <ul>
          <li>Desarrollo web
            <ul>
              <li>HTML5</li>
              <li>CSS3</li>
              <li>JavaScript</li>
            </ul>
          </li>
          <li>Programación
            <ul>
              <li>Python</li>
              <li>Java</li>
            </ul>
          </li>
          <li>Bases de datos
            <ul>
              <li>SQL</li>
            </ul>
          </li>
        </ul>
      </li>
      <li>Blog</li>
      <li>Contacto</li>
    </ul>
  </nav>
</body>
</html>
```

Cada nivel de anidación representa un nivel de la jerarquía del sitio. La lista hija vive dentro del li del padre, nunca fuera. Con CSS después puedes convertir esto en un menú desplegable elegante, pero la base semántica ya está perfecta: un lector de pantalla anunciará correctamente qué opciones pertenecen a cada sección.

Práctica

Aquí es donde el conocimiento se convierte en habilidad. Haz cada ejercicio por tu cuenta antes de ver la solución: el esfuerzo de intentarlo es lo que fija el aprendizaje. Las soluciones están explicadas paso a paso para que compares y entiendas cada decisión.

Ejercicio 1

Enunciado: crea una página HTML con un canvas de 400 píxeles de ancho por 200 de alto, con id 'grafica'. Usando JavaScript, dibuja una gráfica de barras simple con tres barras de colores distintos que representen las ventas de tres meses: enero 120, febrero 80 y marzo 150. Cada barra debe tener 50 píxeles de ancho, estar separada de la siguiente por 20 píxeles, y la altura debe ser proporcional al valor usando una escala donde cada unidad de venta equivale a un píxel de alto. Agrega el texto del mes debajo de cada barra.

Solución paso a paso:

Paso 1: arma la estructura HTML con el canvas, sus dimensiones y su texto alternativo. El id 'grafica' es el puente con JavaScript.

Paso 2: en el script, obtén el canvas y su contexto 2d, como en todos los dibujos de canvas.

Paso 3: calcula las posiciones. Si cada barra mide 50 de ancho y hay 20 de separación, la primera puede empezar en $x = 30$, la segunda en $x = 100$ y la tercera en $x = 170$. Como la y crece hacia abajo, la parte superior de cada barra se calcula restando la altura al fondo de la gráfica. Si el fondo está en $y = 180$, la barra de enero con valor 120 empieza en $y = 60$.

Paso 4: dibuja cada barra con fillRect eligiendo un fillStyle distinto antes de cada una.

Paso 5: escribe el nombre del mes con fillText debajo de cada barra, en $y = 195$.

El código completo queda así:

```

<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Gráfica de barras</title>
</head>
<body>
  <h1>Ventas del trimestre</h1>
  <canvas id='grafica' width='400' height='200'>
    Gráfica de barras: enero 120, febrero 80, marzo 150.
  </canvas>
  <script>
    var pizarra = document.getElementById('grafica');
    var pincel = pizarra.getContext('2d');

    // Fondo de la gráfica
    var fondoY = 180;

    // Datos: mes, valor y color de cada barra
    var datos = [
      { mes: 'Enero', valor: 120, color: 'steelblue' },
      { mes: 'Febrero', valor: 80, color: 'tomato' },
      { mes: 'Marzo', valor: 150, color: 'seagreen' }
    ];

    var x = 30; // posición inicial de la primera barra

    for (var i = 0; i < datos.length; i++) {
      var d = datos[i];
      // La barra empieza arriba: fondo menos la altura del valor
      pincel.fillStyle = d.color;
      pincel.fillRect(x, fondoY - d.valor, 50, d.valor);
      // El nombre del mes debajo de la barra
      pincel.fillStyle = 'black';
      pincel.font = '12px Arial';
      pincel.fillText(d.mes, x, 195);
      // Avanzar a la siguiente posición: ancho más separación
      x = x + 70;
    }
  </script>
</body>
</html>

```

Lo valioso de este ejercicio es que ya no dibujaste a mano cada barra: usaste un arreglo de datos y un ciclo. Esa es la puerta de entrada a las gráficas dinámicas, donde los datos vienen de una base de datos y el dibujo se genera solo. Cambia los valores del arreglo y mira cómo la gráfica se actualiza: estás pensando como programador, no como dibujante.

Ejercicio 2

Enunciado: dibuja en SVG un semáforo. Debe tener un rectángulo vertical oscuro como carcasa, con las esquinas redondeadas usando rx, y tres círculos dentro: rojo arriba, amarillo en medio y verde abajo. Usa viewBox para que escale correctamente. Solo uno de los focos debe verse encendido con color brillante y los otros dos apagados con tonos opacos.

Solución paso a paso:

Paso 1: define el svg con viewBox='0 0 100 260' para tener un mundo interno cómodo, y un tamaño de muestra como width='150' height='390'.

Paso 2: dibuja la carcasa con rect: x='10', y='10', width='80', height='240' y rx='20' para redondear las esquinas. El fill puede ser '#222222'.

Paso 3: coloca los tres círculos centrados en x con cx='50'. Las posiciones verticales: cy='70' para el rojo, cy='130' para el amarillo y cy='190' para el verde, con radio r='25'.

Paso 4: decide cuál está encendido. Por ejemplo el verde: fill='lime' brillante, mientras el rojo lleva fill='#551111' y el amarillo fill='#555511', tonos apagados que sugieren focos apagados.

El código queda así:

```
<svg width='150' height='390' viewBox='0 0 100 260'>
  <!-- Carcasa del semáforo con esquinas redondeadas -->
  <rect x='10' y='10' width='80' height='240' rx='20' fill='#222222' />
  <!-- Foco rojo apagado -->
  <circle cx='50' cy='70' r='25' fill='#551111' />
  <!-- Foco amarillo apagado -->
  <circle cx='50' cy='130' r='25' fill='#555511' />
  <!-- Foco verde encendido -->
  <circle cx='50' cy='190' r='25' fill='lime' />
</svg>
```


Este ejercicio te entrena en el posicionamiento con coordenadas y en el uso de rx para detalles de diseño. Como práctica extra, cambia cuál foco está encendido o agrega un contorno con stroke a la carcasa. Todo lo que aprendas posicionando estas tres formas te sirve para cualquier icono o diagrama futuro.

Ejercicio 3

Enunciado: crea un glosario con dl de al menos seis términos relacionados con HTML5 que hayas aprendido en el curso. Debe incluir: un par simple de término y definición, dos términos que compartan una definición, y un término con dos definiciones. Además, agrega un encabezado h1 con el título del glosario y un párrafo introductorio que explique para qué sirve.

Solución paso a paso:

Paso 1: escribe el h1 y el párrafo de introducción en HTML normal.

Paso 2: abre el dl y agrega los pares simples: por ejemplo 'Etiqueta' con su definición y 'Atributo' con la suya.

Paso 3: agrega los dos términos que comparten definición: 'HTML' y 'HTML5' pueden compartir la definición de lenguaje de marcado, o 'JS' y 'JavaScript' como en el ejemplo de la unidad.

Paso 4: agrega el término con dos acepciones: 'Canvas' puede definirse como la etiqueta que crea la pizarra y también como la técnica de dibujo con JavaScript.

Paso 5: verifica que cada dt y cada dd estén dentro del dl y en el orden correcto.

Ejemplo de solución:

```

<h1>Glosario de HTML5</h1>
<p>
  Este glosario reúne los términos clave del curso.
  Úsalo para repasar antes de cada examen.
</p>
<dl>
  <dt>Etiqueta</dt>
  <dd>Marca que define un elemento en HTML, como p o img.</dd>

  <dt>Atributo</dt>
  <dd>Información extra dentro de una etiqueta que modifica
    su comportamiento, como src o href.</dd>

  <dt>JS</dt>
  <dt>JavaScript</dt>
  <dd>Lenguaje de programación que da interactividad a la web.</dd>

  <dt>Canvas</dt>
  <dd>Etiqueta de HTML5 que crea una pizarra para dibujar.</dd>
  <dd>Técnica de dibujo con píxeles controlada por JavaScript.</dd>

  <dt>SVG</dt>
  <dd>Gráficos vectoriales que se dibujan con código y no se pixelan.</dd>

  <dt>Semántica</dt>
  <dd>Uso de etiquetas que describen el significado del contenido.</dd>
</dl>

```

El valor de este ejercicio está en pensar qué contenido merece una lista de definición. Cada vez que organices información en pares de concepto y explicación estarás usando la herramienta correcta en lugar de improvisar con párrafos o tablas.

Ejercicio 4

Enunciado: construye con listas anidadas el índice de un curso ficticio de tres módulos. Cada módulo es una lista ordenada con `type='I'` para números romanos, y dentro de cada módulo hay una lista sin orden con los temas. El módulo 2 debe empezar su numeración de temas desde la letra C usando `start`. Asegúrate de que cada lista anidada quede dentro del `li` correspondiente.

Solución paso a paso:

Paso 1: crea el ol principal con type='I' para que los módulos se numeren I, II, III.

Paso 2: dentro de cada li del módulo, anida un ul con los temas. Recuerda: el ul va dentro del li, antes de su cierre.

Paso 3: en el módulo 2, usa un ol con type='a' y start='3' para que los temas se listen como c, d, e, demostrando que dominas la continuación de numeración.

Paso 4: revisa la estructura completa contando aperturas y cierres.

Código de solución:

```
<h1>Índice del curso</h1>
<ol type='I'>
  <li>Fundamentos
    <ul>
      <li>Qué es la web</li>
      <li>Cómo funciona un navegador</li>
    </ul>
  </li>
  <li>Etiquetas esenciales
    <ol type='a' start='3'>
      <li>Texto y encabezados</li>
      <li>Enlaces e imágenes</li>
    </ol>
  </li>
  <li>Proyecto final
    <ul>
      <li>Planeación del sitio</li>
      <li>Publicación en internet</li>
    </ul>
  </li>
</ol>
```

Fíjate que mezclar ol y ul anidados es válido y a veces es justo lo que el contenido pide: los módulos tienen un orden, pero los temas dentro de un módulo pueden ser una simple colección. El start='3' con type='a' produce c y d, simulando que la lista continúa algo anterior, como un temario dividido en partes.

Ejercicio 5

Enunciado: reto integrador. Crea una página completa que combine los tres temas de la unidad: un encabezado con el título, un canvas de 480 por 200 donde dibujes con JavaScript tres círculos de colores que representen un semáforo horizontal apagado o encendido a tu elección, un logo sencillo en SVG de 120 por 120 con viewBox que incluya al menos un círculo y un rectángulo, y un glosario con dl de tres términos de esta unidad: canvas, SVG y lista de definición. Todo en un solo archivo HTML válido.

Solución paso a paso:

Paso 1: arma el esqueleto HTML con doctype, head con charset y title, y body.

Paso 2: agrega el h1 y una breve introducción en un párrafo.

Paso 3: inserta el canvas con id, dimensiones y texto alternativo. En el script dibuja los tres círculos con arc: calcula sus centros para repartirlos en horizontal, por ejemplo x en 90, 240 y 390, todos con cy='100' y radio 45. Enciende uno con color brillante y apaga los otros dos.

Paso 4: agrega el SVG del logo con viewBox='0 0 120 120': un círculo de fondo y un rectángulo al centro como símbolo, más el texto si quieres.

Paso 5: cierra con el dl de tres términos, cada uno con su definición escrita con tus propias palabras, que es la mejor prueba de que entendiste.

Paso 6: abre el archivo en el navegador y verifica las tres partes: el semáforo dibujado, el logo nítido y el glosario con formato de diccionario.

Código de solución:

```
<!DOCTYPE html>
<html lang='es'>
<head>
  <meta charset='UTF-8'>
  <title>Reto integrador unidad 7</title>
</head>
<body>
  <h1>Lo que aprendí en la unidad 7</h1>
  <p>Esta página combina canvas, SVG y listas de definición.</p>

  <h2>Semáforo en canvas</h2>
  <canvas id='semaforo' width='480' height='200'>
    Semáforo horizontal con el foco verde encendido.
  </canvas>
```

```

<h2>Mi logo en SVG</h2>
<svg width='120' height='120' viewBox='0 0 120 120'>
  <circle cx='60' cy='60' r='55' fill='#1e5aa8' />
  <rect x='40' y='40' width='40' height='40' fill='white' />
</svg>

<h2>Glosario de la unidad</h2>
<dl>
  <dt>Canvas</dt>
  <dd>Pizarra de HTML5 donde JavaScript dibuja con píxeles.
    Ideal para juegos y gráficas dinámicas.</dd>
  <dt>SVG</dt>
  <dd>Gráficos vectoriales dibujados con etiquetas que nunca
    se pixelan. Perfectos para logos e iconos.</dd>
  <dt>Lista de definición</dt>
  <dd>Estructura con dl, dt y dd para emparejar términos
    con sus explicaciones, como un diccionario.</dd>
</dl>

<script>
  var pizarra = document.getElementById('semaforo');
  var pincel = pizarra.getContext('2d');

  // Carcasa horizontal del semáforo
  pincel.fillStyle = '#222222';
  pincel.fillRect(20, 50, 440, 100);

  // Focos: rojo y amarillo apagados, verde encendido
  var focos = [
    { x: 90, color: '#551111' },
    { x: 240, color: '#555511' },
    { x: 390, color: 'lime' }
  ];

  for (var i = 0; i < focos.length; i++) {
    pincel.fillStyle = focos[i].color;
    pincel.beginPath();
    pincel.arc(focos[i].x, 100, 40, 0, Math.PI * 2);
    pincel.fill();
  }
</script>
</body>
</html>

```

Si lograste este ejercicio sin copiar la solución, ya dominas lo esencial de la unidad: sabes preparar un canvas y dibujar en él con lógica de programación, sabes construir un SVG con viewBox y formas básicas, y sabes estructurar definiciones con dl. Ese es exactamente el nivel que esta unidad quería llevarte. Guárdalo en tu carpeta de práctica: es una gran pieza para tu portafolio.

Proyecto de la unidad: página de presentación de un curso

Para cerrar con broche de oro, aquí tienes un proyecto que junta todo lo aprendido en un solo archivo. La idea es crear la página de presentación de un curso ficticio, como las que verás en lucasegaacademy.com, usando canvas, SVG y listas de definición donde corresponde. No es un ejercicio con solución única: es tu creación, y el plan de abajo es tu mapa.

El plan paso a paso:

- Paso 1: arma el esqueleto HTML con un encabezado que tenga el título del curso y un menú de navegación hecho con una lista anidada: secciones principales y, dentro de la sección de temario, los módulos como sublista.
- Paso 2: crea un banner con canvas de 600 por 150 píxeles donde dibujes con JavaScript el nombre del curso con fillText sobre un fondo de rectángulos de colores. Es tu cartelera digital.
- Paso 3: diseña un logo sencillo en SVG con viewBox que combine al menos tres formas distintas: por ejemplo un círculo de fondo, un polygon como símbolo y texto con el nombre del curso.
- Paso 4: escribe la sección de preguntas frecuentes con dl, con al menos cuatro pares de pregunta y respuesta reales sobre tu curso ficticio.
- Paso 5: agrega un glosario con dl de cinco términos del curso, incluyendo al menos un caso de dos términos que compartan definición.
- Paso 6: revisa tu página con la lista de verificación: ¿el canvas tiene dimensiones e id? ¿el SVG tiene viewBox? ¿las listas anidadas están dentro de sus li? ¿cada dt y dd está dentro de un dl? ¿el texto alternativo del canvas describe el dibujo?

Cuando termines, vas a tener una página que demuestra las tres habilidades de la unidad en un contexto real. Guárdala en tu portafolio con orgullo: pocos principiantes pueden decir que dibujan con código y estructuran información como un profesional. Y si te animas, dale tu toque personal: cambia los colores, agrega más formas al logo o una animación sencilla en el banner con requestAnimationFrame. Esta unidad te dio las herramientas; el estilo lo pones tú.

Resumen de la unidad

- Canvas es la etiqueta de HTML5 que crea una pizarra digital: HTML pone el espacio con `<canvas>` y JavaScript dibuja en él con el contexto 2d.
- Todo canvas debe declarar `width` y `height` en la etiqueta, llevar un `id` para que JavaScript lo encuentre y contener texto alternativo para navegadores que no lo soporten.
- En canvas el origen de coordenadas (0, 0) está en la esquina superior izquierda: la `x` crece a la derecha y la `y` crece hacia abajo.
- Las órdenes básicas de dibujo son `fillRect` para rectángulos, `beginPath` con `moveTo` y `lineTo` para trazos, `arc` para círculos, y `fillText` para texto, con `fillStyle` y `strokeStyle` para los colores.
- Canvas es la base para juegos, animaciones y gráficas dinámicas, porque JavaScript puede borrar y redibujar la pizarra muchas veces por segundo.
- Lo dibujado en canvas se vuelve píxeles: no se puede mover por partes después ni lo leen los lectores de pantalla, así que no pongas ahí información importante solo como texto.
- SVG son gráficos vectoriales: describen formas con matemáticas en lugar de píxeles, por eso nunca se pixelan al agrandarlos.
- Las formas principales de SVG son `circle`, `rect`, `line`, `ellipse`, `polygon` y `path`, y se controlan con `fill`, `stroke` y `stroke-width`.
- El `viewBox` define el sistema de coordenadas interno del SVG y es lo que permite escalarlo a cualquier tamaño sin deformarse: inclúyelo siempre.
- Usa SVG para logos, iconos y diagramas; usa canvas para juegos y gráficas en movimiento; usa `img` con JPG o PNG para fotografías.
- La lista de definición usa `dl` como contenedor, `dt` para cada término y `dd` para su definición: es la estructura correcta para glosarios, preguntas frecuentes y fichas de datos.
- Las listas anidadas van siempre dentro del elemento `li` padre, y puedes mezclar `ul` y `ol` según lo pida el contenido.
- Los atributos `type` y `start` de las listas ordenadas controlan el estilo de numeración y desde dónde empieza la cuenta.
- Errores que ya no vas a cometer: canvas sin dimensiones, olvidar `beginPath`, SVG sin `viewBox`, usar SVG para fotos y anidar listas fuera del `li`.

- Con degradados, transparencias y sombras en canvas tus dibujos pasan de planos a profesionales: `createLinearGradient`, `globalAlpha` y las propiedades de sombra son tus aliadas.
- La animación en canvas sigue un ciclo simple: borrar con `clearRect`, dibujar, mover las variables de estado y repetir con `requestAnimationFrame`.
- En SVG, la etiqueta `g` agrupa formas para moverlas y darles estilo en conjunto, y el texto sigue siendo texto real: seleccionable y accesible.
- Las preguntas frecuentes y las fichas de datos son el uso más común de `dl` en sitios reales: cada pregunta o etiqueta es un `dt` y cada respuesta o valor es un `dd`.

Felicidades por terminar la unidad 7. Ya no solo estructuras páginas: ahora las ilustras con gráficos hechos por ti y organizas la información con las listas correctas. En la siguiente unidad seguirás sumando herramientas, y cada vez que mires un logo nítido o una gráfica animada en la web vas a saber exactamente cómo se hizo por dentro. Sigue practicando, sigue dibujando con código y nos vemos en la unidad 8.

UNIDAD 8: Precisión, accesibilidad y buenas prácticas

Felicidades por llegar hasta aquí. Ya no eres el mismo que empezó este curso: sabes estructurar un documento con encabezados y párrafos, insertar imágenes y enlaces, armar tablas que se entienden, construir formularios que capturan datos y darle formato al texto con las etiquetas correctas. Tienes las herramientas para levantar una página web completa. Pero aquí viene una verdad que distingue a los aficionados de los profesionales: que una página "se vea bien en tu pantalla" no significa que esté bien hecha. Esta unidad trata de todo lo que no se ve a simple vista y que, sin embargo, define la calidad real de tu trabajo.

En esta unidad vas a dominar tres temas que parecen detalles pequeños pero que en la práctica lo cambian todo. Primero, la precisión con entidades y símbolos HTML: aprenderás a escribir caracteres especiales como menor que, mayor que, el ampersand y símbolos como los de derechos de autor o el euro, y sobre todo aprenderás a mostrar código como texto sin que el navegador lo ejecute, algo indispensable si algún día escribes tutoriales, documentación o un blog técnico como este. Segundo, la accesibilidad web: descubrirás qué significa construir una web para todos, incluyendo a las personas que navegan con lector de pantalla, que usan solo el teclado o que no distinguen bien los colores, y verás que la accesibilidad no es un adorno extra sino parte del HTML bien escrito. Tercero, las buenas prácticas profesionales: los hábitos de los programadores que trabajan en serio, desde indentar tu código y usar minúsculas hasta comentar con inteligencia, preferir etiquetas semánticas antes que div, no repetir identificadores, validar tu HTML con el validador de la W3C y ponerle nombres sensatos a tus archivos.

¿Por qué importa tanto la precisión? Porque en HTML un solo carácter mal colocado puede cambiar por completo lo que el navegador entiende. El signo de menor que < no es un adorno: para el navegador es una orden que dice "aquí empieza una etiqueta". Si quieres mostrar ese signo como texto normal, por ejemplo para enseñarle a alguien cómo se escribe un párrafo, necesitas un mecanismo especial para decirle al navegador: "oye, esto no es una instrucción, es texto". Ese mecanismo son las entidades HTML, y dominarlas te evita errores misteriosos donde tu página muestra cosas raras o directamente se rompe. Además, la precisión incluye escribir correctamente símbolos como ©, ® o €, que dan a tus páginas un acabado profesional y evitan esos cuadritos extraños o signos de interrogación que aparecen cuando los caracteres no se codifican bien.

¿Por qué importa la accesibilidad? Piensa en esto: hay millones de personas en el mundo que usan la web de formas distintas a la tuya. Alguien ciego navega con un lector de pantalla que le lee tu página en voz alta; si tus imágenes no tienen texto alternativo, para esa persona esas imágenes simplemente no existen. Alguien que no puede usar el mouse navega solo con el teclado; si tu página no tiene un orden lógico ni un foco visible, esa persona se pierde. La accesibilidad es, antes que nada, un asunto de respeto: la web nació para ser universal, para todos, sin excepciones. Pero además tiene razones muy prácticas: en muchos países es una obligación legal, amplía tu audiencia potencial a millones de personas más y hasta mejora tu posicionamiento en buscadores, porque los motores de búsqueda premian el HTML semántico y bien estructurado. Y aquí está el secreto mejor guardado: la accesibilidad no se logra con trucos complicados, se logra escribiendo HTML correcto. El alt descriptivo, el label asociado a su input, la jerarquía lógica de encabezados: todo eso es HTML básico bien aplicado.

¿Por qué importan las buenas prácticas? Porque el código se lee muchas más veces de las que se escribe. Tú mismo vas a volver a tus archivos dentro de tres meses y vas a necesitar entender qué hiciste; si tu código está desordenado, sin indentar, con mayúsculas mezcladas y sin un solo comentario, vas a perder horas descifrando tu propio trabajo. Y cuando trabajes con otros programadores, tu código es tu carta de presentación: un HTML limpio, indentado, comentado y validado dice "esta persona sabe lo que hace" antes de que abras la boca. Las buenas prácticas son como la higiene: nadie te aplaude por lavarte las manos, pero todo funciona mejor cuando lo haces, y cuando no lo haces, tarde o temprano alguien lo nota.

Esta unidad encaja en el curso como la etapa de pulido después de la construcción. En las unidades anteriores levantaste los muros: estructura del documento, texto, imágenes, enlaces, tablas y formularios. Ahora toca dejar la casa habitable, segura y presentable: las entidades son como las cerraduras bien instaladas, que hacen que cada pieza quede exactamente en su lugar; la accesibilidad son las rampas y los pasamanos, que permiten que todos entren y se muevan sin barreras; y las buenas prácticas son la limpieza y el orden de las instalaciones, lo que hace que la casa se mantenga en pie y sea fácil de reparar durante años. Sin esta unidad, todo lo que aprendiste antes funciona a medias: una página sin charset declarado muestra tildes rotas, un formulario sin labels excluye usuarios, un código sin validar se ve distinto en cada navegador.

Lo que sigue después de esta unidad se construye directamente sobre estos hábitos. Cuando llegues a temas más avanzados y al proyecto final del curso, vas a agradecer cada buena práctica que interiorices hoy: el validador de la W3C te va a ahorrar horas de depuración, la semántica te va a dar puntos extra en accesibilidad y SEO sin esfuerzo adicional, y la costumbre de comentar e indentar va a hacer que tu proyecto final sea un código del que te

sientas orgulloso de mostrar. Los profesionales no son los que nunca cometen errores, son los que tienen sistemas para detectarlos: el checklist de calidad que verás al final de esta unidad es exactamente eso, tu sistema personal de control de calidad.

No te voy a mentir: esta unidad tiene menos "efectos espectaculares" que otras. Nadie va a decir "wow" al ver un `<` bien escrito. Pero créeme cuando te digo que estos son los temas que separan al que cobra como profesional del que sigue haciendo páginas de hobby. Cada detalle que aprendas aquí es una de esas cosas que los reclutadores y los clientes notan sin saber exactamente por qué: "este código se ve profesional". Al terminar esta unidad vas a escribir HTML preciso, inclusivo y limpio, vas a tener tu propio checklist de calidad antes de publicar, y vas a entender por qué los mejores desarrolladores del mundo son obsesivos con cosas que parecen insignificantes. Échale ganas: el diablo está en los detalles, y a partir de hoy, los detalles trabajan para ti.

Explicación detallada

Por qué existen las entidades HTML

Imagina que estás escribiendo una carta y de pronto necesitas escribir la palabra "punto" sin que el lector piense que terminaste la oración. Suena absurdo, pero eso es exactamente lo que le pasa al navegador con ciertos caracteres. En HTML hay caracteres reservados, es decir, caracteres que el navegador interpreta como instrucciones y no como texto normal. Los más importantes son tres: el signo de menor que `<`, el signo de mayor que `>` y el ampersand `&`. Cuando el navegador ve un `<`, no piensa "el usuario quiere mostrar un signo de menor que", piensa "aquí empieza una etiqueta, voy a interpretarla como código". Y eso está muy bien cuando efectivamente quieres escribir código, pero es un problema cuando quieres mostrar esos signos como texto visible en tu página.

Aquí es donde entran las entidades HTML. Una entidad es un código especial que le dice al navegador: "muestra este carácter como texto, no lo interpretes como instrucción". Es como una clave secreta entre tú y el navegador: tú escribes `<` y el navegador muestra `<` en la pantalla, sin tratar de convertirlo en una etiqueta. Piensa en las entidades como el lenguaje en clave de los espías en las películas: el mensaje viaja disfrazado para que solo quien conoce la clave lo entienda correctamente. Tú escribes el código secreto en tu archivo HTML y el navegador, que conoce la clave, muestra el carácter real al visitante.

Todas las entidades comparten el mismo formato: empiezan con un ampersand `&`, siguen con un nombre o un número, y terminan con punto y coma;. Ese punto y coma final no es

decorativo: es obligatorio. Si escribes < sin el punto y coma, el navegador no reconoce la entidad y muestra el texto tal cual, con todo y ampersand, lo cual se ve terrible y poco profesional. Grábate esto: entidad sin punto y coma es como teléfono sin marcar el último dígito, simplemente no conecta.

Un ejemplo clásico donde las entidades salvan el día: quieres escribir un tutorial que enseñe cómo se hace un párrafo en HTML. Necesitas mostrar el texto `<p>Hola mundo</p>` en la pantalla. Si lo escribes tal cual en tu archivo, el navegador lo interpreta como un párrafo real y el visitante solo ve "Hola mundo", sin las etiquetas. Tu tutorial queda inútil porque justo lo que querías enseñar desapareció. La solución es escribir `<p>Hola mundo</p>`, y entonces el navegador muestra `<p>Hola mundo</p>` como texto visible. Más adelante en esta unidad verás un ejemplo completo de una página de tutorial construida con esta técnica, que es la base de sitios educativos como esta misma academia.

El ampersand merece una mención especial porque es el más traicionero de los tres. Como todas las entidades empiezan con &, el navegador siempre está alerta cuando ve uno: intenta leer lo que sigue como si fuera el nombre de una entidad. Si escribes "Q&A" directamente, algunos navegadores lo muestran bien por casualidad, pero tu HTML es inválido y en otros navegadores o contextos puede mostrarse mal. La forma correcta y profesional es escribir "Q&A". Lo mismo pasa con nombres de empresas como "AT&T", que debe escribirse "AT&T", o con textos como "términos & condiciones", que debe ser "términos & condiciones". Cada vez que escribas un & que sea parte del texto y no el inicio de una entidad, escápalo como &. Es una de esas reglas que separan al profesional del aficionado: el aficionado confía en la suerte, el profesional escribe & siempre.

Las entidades que no te pueden faltar

Vamos a conocer a las entidades más útiles, esas que vas a usar una y otra vez en tu vida como desarrollador. Memorízalas como te memorizaste las tablas de multiplicar: son pocas y te acompañan siempre.

- ` ` es el espacio inseparable. Se ve como un espacio normal, pero le dice al navegador: "no separes estas dos palabras en un salto de línea". Úsalo cuando dos elementos deben permanecer juntos sí o sí, por ejemplo "`10 km`", "`Sr. García`" o "`$ 500`". Sin el espacio inseparable, el navegador podría dejar el "10" al final de una línea y el "km" al inicio de la siguiente, lo cual se ve mal. Ojo con el error clásico de principiante: usar ` ` repetidos para indentar o para separar elementos visualmente, como `     `. Eso es un mal hábito heredado de los años noventa; para espacios visuales existe CSS con sus márgenes y rellenos. El ` ` es para unir palabras, no para empujar cosas.
- `<` es el signo de menor que `<`. Lo usas cada vez que necesitas mostrar una etiqueta HTML como texto, como en tutoriales, documentación o ejemplos de código. Es la entidad más famosa y la que más vas a escribir.
- `>` es el signo de mayor que `>`. Siempre acompaña a `<` cuando muestras etiquetas completas: `<p>` muestra `<p>`. También sirve para mostrar comparaciones matemáticas como texto, por ejemplo "`5 > 3`".
- `&` es el ampersand `&`. Ya vimos por qué es traicionero: úsalo siempre que el `&` forme parte del texto normal y no sea el inicio de otra entidad.
- `"` son las comillas dobles `"`. En texto normal no necesitas escaparlas, pero son útiles cuando escribes comillas dentro de un atributo que ya usa comillas dobles, o cuando generas código de ejemplo muy anidado. Por ejemplo, si muestras un fragmento de código que contiene atributos con comillas, `"` te ayuda a que todo quede claro.
- `©` es el símbolo de derechos de autor `©`. Todo sitio profesional lo usa en su pie de página: `<p>© 2026 Lucas EGA Academy. Todos los derechos reservados.</p>` se muestra como "`© 2026 Lucas Ega Academy. Todos los derechos reservados.`" Es un detalle pequeño que grita profesionalismo.
- `®` es el símbolo de marca registrada `®` y `™` es el de marca comercial `™`. Los usas cuando mencionas marcas: "`SuperCurso®`" o "`MiProducto™`".
- `€` es el símbolo del euro, y también existen `£` para la libra `£`, `¥` para el yen `¥` y `¢` para el centavo `¢`. Son indispensables en tiendas en línea y sitios de finanzas. Escribir el símbolo directamente desde el teclado a veces funciona, pero la entidad garantiza que se muestre bien en cualquier navegador y configuración.
- Las entidades de vocales acentuadas como `á` para `á` o `ñ` para `ñ` existieron porque antes los documentos no siempre declaraban bien su codificación. Hoy, con la declaración de charset UTF-8 que verás en la siguiente subsección, puedes escribir tildes

y eñes directamente en tu código sin problema. Menciono estas entidades para que las reconozcas si las ves en código viejo, pero en código nuevo escribe los acentos directamente: es más legible y es el estándar actual.

¿Cómo encontrar más entidades? Existen cientos, para flechas, símbolos matemáticos, letras griegas y más. Busca "lista de entidades HTML" cuando necesites alguna en especial. Pero con las de esta lista cubres el 95 por ciento de lo que vas a necesitar en tu carrera.

Cómo mostrar código como texto sin que se ejecute

Este es el superpoder práctico de las entidades y merece su propia explicación paso a paso, porque es la técnica detrás de cada tutorial, cada blog de programación y cada documentación técnica que existe en la web, incluyendo los materiales de esta academia.

El problema, recapitulando: quieres que el visitante vea el código, no que el navegador lo ejecute. Si escribes `<strong>importante</strong>` directamente, el navegador muestra la palabra "importante" en negritas y las etiquetas desaparecen. Para mostrar el código tal cual, debes escapar los tres caracteres reservados: cada `<` se convierte en `<`, cada `>` en `>`; y cada `&` en `&`. El orden importa cuando lo haces a mano: primero escapa los ampersands que ya existan en el texto, y después los signos de menor y mayor que. Si lo haces al revés, puedes terminar escapando dos veces y mostrando `&lt;`; en pantalla, que se ve como `"<"` literal, un error muy común y muy visible.

Además de escapar, necesitas dos etiquetas ayudantes: `<pre>` y `<code>`. La etiqueta `<pre>` le dice al navegador que conserve los espacios y los saltos de línea tal como los escribiste, porque normalmente HTML colapsa los espacios múltiples en uno solo y ignora los saltos de línea de tu código fuente. La etiqueta `<code>` le dice al navegador, y a los lectores de pantalla, que el contenido es código de computadora, y por defecto lo muestra con una fuente monoespaciada que facilita la lectura. La combinación estándar es anidarlas así: `<pre><code>tu código escapado aquí</code></pre>`. El `<pre>` cuida el formato y el `<code>` cuida el significado.

Veamos la transformación completa con un ejemplo. Supón que quieres mostrar este fragmento: `<p class="saludo">Hola mundo</p>`. En tu archivo HTML debes escribir: `<p class="saludo">Hola mundo</p>`, envuelto en `<pre><code>`. El visitante verá exactamente `<p class="saludo">Hola mundo</p>` con su indentación y sus saltos de línea, como código de ejemplo. En la sección de ejemplos paso a paso de esta unidad encontrarás una página de tutorial completa construida con esta técnica, lista para que la estudies y la adaptes.

Un error común es escapar solo el signo de menor que y olvidar el ampersand. Por ejemplo, si tu código de ejemplo contiene "a && b" y solo escapas los < y >, el && queda crudo y el navegador intentará interpretarlo como entidades, mostrando basura o nada. La regla de oro: dentro de un bloque de código visible, escapa los tres, siempre. Otro error es olvidar cerrar el <pre> o el <code>, lo cual hace que todo el resto de tu página se muestre con formato de código. Y un consejo de profesional: si usas un editor moderno o un sistema de gestión de contenido, muchos tienen botones o atajos para escapar automáticamente; úsalos cuando existan, pero asegúrate de entender qué hacen por debajo, porque algún día tendrás que hacerlo a mano.

El atributo charset UTF-8 y su relación con los símbolos

Muy bien, ya sabes escribir símbolos con entidades. Pero hay una pregunta natural: ¿por qué no escribir directamente la ñ, las tildes y el símbolo de euro en mi archivo? La respuesta es que sí puedes, siempre y cuando le digas al navegador qué codificación de caracteres estás usando. Esa declaración se hace con la etiqueta <meta charset="UTF-8"> dentro del <head> de tu documento, y es una de las líneas más importantes de todo tu HTML.

Para entenderlo, piensa en la codificación como el idioma secreto con el que tu archivo guarda las letras. Tu archivo no guarda "letras" como tales, guarda números, y existe una tabla que dice qué número corresponde a cada letra. UTF-8 es la tabla universal moderna: incluye las letras del español con sus tildes, la ñ, los símbolos de monedas como el euro, y prácticamente todos los caracteres de todos los idiomas del mundo. Cuando pones <meta charset="UTF-8">, le estás diciendo al navegador: "oye, interpreta los números de este archivo usando la tabla UTF-8". Si no se lo dices, el navegador adivina, y a veces adivina mal.

¿Y qué pasa cuando adivina mal? Ocurre el famoso texto mojibake, esos caracteres rotos que todos hemos visto: "programaciÃ³n" en vez de "programación", o "niÃ±o" en vez de "niño". Eso pasa porque el archivo se guardó en UTF-8 pero el navegador lo interpretó con otra tabla. Es uno de los errores más vergonzosos en una página profesional, porque grita "a esto no le pusieron atención", y se evita con una sola línea de código. Por eso la regla es absoluta: todo documento HTML debe empezar su <head> con <meta charset="UTF-8">, colocada lo antes posible, idealmente como la primera etiqueta dentro del <head>. Los navegadores solo miran los primeros 1024 bytes del documento para detectar la codificación, así que si la pones al final del <head> después de muchos scripts o estilos, puede que no la vean a tiempo.

Entonces, ¿para qué sirven las entidades si ya tengo UTF-8? Para los caracteres reservados: <, > y & siempre necesitarán entidades cuando quieras mostrarlos como texto, porque UTF-8 no cambia el hecho de que el navegador los interpreta como instrucciones. Las entidades de

símbolos como © o € siguen siendo útiles cuando tu teclado no tiene el símbolo, cuando quieres máxima compatibilidad con sistemas viejos, o cuando prefieres que tu código fuente sea 100 por ciento ASCII por alguna razón técnica. Pero para tildes, eñes y diéresis del día a día, escribe directamente los caracteres con UTF-8 declarado: tu código queda más limpio y legible. En resumen: UTF-8 para el texto normal en español, entidades para los caracteres reservados y para símbolos especiales cuando te convenga. Con estas dos herramientas, tu página mostrará cada carácter exactamente como lo planeaste, en cualquier navegador del mundo.

Qué es la accesibilidad web y por qué importa

La accesibilidad web significa que cualquier persona, sin importar sus capacidades físicas, sensoriales o cognitivas, pueda percibir, entender, navegar e interactuar con tu sitio. Suena como una definición de diccionario, así que vamos a ponerle rostros. María es ciega y navega con un lector de pantalla, un programa que le lee en voz alta todo el contenido de la página; si tus imágenes no tienen texto alternativo, para María esas imágenes no existen. Don Roberto tiene 72 años, la vista cansada y el pulso tembloroso; necesita textos con buen contraste y botones grandes que pueda activar sin precisión quirúrgica. Ana tuvo un accidente y temporalmente solo puede usar una mano, así que navega todo con el teclado; si tu página no tiene un orden lógico ni muestra dónde está el foco, Ana se pierde. Y Luis es daltónico: si tu formulario marca los errores solo con color rojo, él no se entera de qué salió mal.

La web nació con la promesa de ser universal. Su creador soñó con un espacio al que todos pudieran contribuir y del que todos pudieran beneficiarse, sin barreras. Cada vez que publicas una página inaccesible, rompes un poquito esa promesa. Pero no te estoy pidiendo caridad: la accesibilidad también es una decisión inteligente por tres razones muy terrenales. Primera, la legal: en muchos países existen leyes que obligan a que los sitios de gobierno y empresas cumplan estándares de accesibilidad, y las demandas por sitios inaccesibles crecen cada año; construir accesible desde el inicio te evita problemas caros después. Segunda, el negocio: alrededor del 15 por ciento de la población mundial vive con alguna discapacidad, eso son más de mil millones de personas; excluirlas es como cerrar la puerta de tu tienda a uno de cada siete clientes que pasan por la banqueta. Tercera, el posicionamiento: los motores de búsqueda son, en cierto sentido, usuarios ciegos que no ven tus imágenes ni tus colores, solo leen tu código; un HTML semántico, con buenos textos alternativos y jerarquía clara de encabezados, es exactamente lo que Google premia.

Y aquí viene la mejor noticia de toda esta subsección: la accesibilidad no requiere magia ni herramientas caras. El 80 por ciento de la accesibilidad se logra con HTML correcto, el mismo

HTML que ya estás aprendiendo. Un alt descriptivo, un label asociado a su input, un solo h1 seguido de h2 en orden, el atributo lang en la etiqueta html: todo eso es HTML básico bien aplicado. La accesibilidad no es una capa extra que se pega al final como barniz; es el resultado natural de escribir HTML como se debe. Por eso esta unidad existe: porque ya tienes las piezas, y ahora vas a aprender a usarlas con conciencia.

La analogía perfecta son las rampas en los edificios. Cuando un arquitecto diseña un edificio solo con escaleras, está diciendo sin palabras: "las personas en silla de ruedas no son bienvenidas aquí". Poner una rampa después, como parche, sale caro y queda mal. En cambio, si la rampa se diseña desde el plano, el edificio queda mejor para todos: la usan las sillas de ruedas, sí, pero también las carriolas, los repartidores con diablitos y los viajeros con maletas. Con la web pasa igual: los subtítulos que ayudan a las personas sordas también ayudan a quien ve tu video en el camión sin audífonos; el buen contraste que ayuda a la vista cansada también ayuda a quien lee tu página bajo el sol. Diseñar accesible desde el inicio beneficia a todos y cuesta menos que remendar después.

El atributo alt: los ojos de quien no puede ver

El atributo alt de la etiqueta `<img>` es la herramienta de accesibilidad más famosa y la más mal usada de toda la web. Su función es sencilla: proporcionar un texto alternativo que describa la imagen para quienes no pueden verla. Ese texto lo lee el lector de pantalla en voz alta, y también se muestra en la página si la imagen no se carga por un error de conexión o una ruta rota. En otras palabras, el alt son los ojos prestados de tu imagen.

Escribir un buen alt es un arte con reglas claras. Primera regla: describe el contenido y la función de la imagen, no su apariencia técnica. Un mal alt dice "imagen" o "foto", que es como no decir nada: el lector de pantalla ya anuncia que se trata de una imagen, así que estás desperdiciando palabras. Segunda regla: sé conciso pero completo, normalmente una frase basta. Tercera regla: si la imagen es un enlace o un botón, el alt debe describir la acción o el destino, no la imagen misma.

Veamos ejemplos concretos. Supón que tienes la foto del profesor frente al pizarrón en la página de un curso. Un mal alt sería `alt="imagen"` o `alt="foto1.jpg"`. Un alt mediocre sería `alt="foto del profesor"`. Un buen alt sería `alt="Profesor explicando etiquetas HTML frente a un pizarrón verde"`. ¿Ves la diferencia? El buen alt le entrega a la persona ciega la misma información esencial que la imagen le entrega a quien sí ve. Otro caso: el logotipo de tu academia en el encabezado, que además es un enlace a la página de inicio. El alt no debe decir "logotipo", debe decir hacia dónde lleva: `alt="Lucas EGA Academy - inicio"`. Y si el botón de búsqueda de tu sitio es solo una lupita sin texto, el alt debe decir `alt="Buscar"`.

Ahora, el caso especial que casi nadie conoce: las imágenes puramente decorativas. Esas líneas, sombras, texturas de fondo o iconos que solo adornan y no aportan información deben llevar `alt=""` con las comillas vacías. Ese alt vacío es una instrucción explícita al lector de pantalla: "ignora esta imagen, no dice nada importante". Si en cambio le pones `alt="decoración"` o `alt="adorno"`, el lector lo anunciará en voz alta y la persona tendrá que escuchar basura irrelevante en cada página. Y si omites el alt por completo, muchos lectores anuncian el nombre del archivo, algo como "imagen banner final 2 punto jpg", que es todavía peor. Regla de oro: toda `<img>` lleva alt siempre; si la imagen informa, describe; si decora, alt vacío.

Para imágenes complejas como gráficas de barras o diagramas, un alt corto no alcanza. La estrategia profesional es doble: un alt breve que diga de qué trata la gráfica, por ejemplo `alt="Gráfica de barras: crecimiento de estudiantes 2024 a 2026"`, más una descripción detallada en el texto cercano a la imagen o dentro de un `<figure>` con `<figcaption>`. Así la persona ciega recibe primero el resumen y puede decidir si quiere el detalle, igual que una persona que ve primero le echa un vistazo a la gráfica y luego lee la explicación si le interesa.

Los errores más comunes con el alt son fáciles de evitar una vez que los conoces: omitirlo por completo, escribir "imagen de..." redundante, poner el nombre del archivo como alt, describir imágenes decorativas en vez de vaciarlas, y escribir párrafos enteros donde basta una frase. Hazte el hábito de preguntarte con cada imagen: "si esta imagen desapareciera, ¿qué información perdería el visitante?". La respuesta a esa pregunta es tu alt.

label y aria-label: formularios para todos

Los formularios son el lugar donde la accesibilidad se gana o se pierde, porque un formulario inaccesible no solo es incómodo: directamente impide que la persona complete la tarea, ya sea registrarse, comprar o contactarte. Y la buena noticia es que la base de un formulario accesible es una sola etiqueta que ya conoces: `<label>`.

La etiqueta `<label>` asocia un texto descriptivo con un campo del formulario. La asociación se hace con dos atributos que trabajan en pareja: el label lleva `for="email"` y el input lleva `id="email"`, con el mismo valor en ambos. Cuando están correctamente asociados ocurren dos cosas mágicas. Primera: si el usuario hace clic en el texto "Correo electrónico", el cursor salta automáticamente al campo correspondiente; esto ayuda a todos, pero especialmente a personas con pulso tembloroso o en pantallas táctiles pequeñas, porque el área clicable se vuelve mucho más grande. Segunda: el lector de pantalla anuncia el label cuando el usuario llega al campo, así la persona ciega sabe qué dato debe escribir. Sin label asociado, el lector solo dice "campo de texto" y la persona tiene que adivinar qué va ahí.

El error más común y más dañino en formularios modernos es usar el placeholder como única etiqueta. Ya sabes que placeholder muestra un texto de ejemplo dentro del campo que desaparece cuando el usuario empieza a escribir. El problema es triple: cuando el texto desaparece, el usuario ya no puede verificar qué campo es ese, lo cual afecta a personas con problemas de memoria o a cualquiera que se distraiga a mitad del formulario; muchos lectores de pantalla no anuncian el placeholder de forma confiable; y el contraste del placeholder suele ser bajo, difícil de leer. La regla profesional es clara: el placeholder es una ayuda opcional con un ejemplo de formato, nunca el sustituto del label. Un formulario correcto tiene `<label for="nombre">Nombre completo</label>` visible siempre, y opcionalmente `<input placeholder="Ej. María García">` como ejemplo.

Cuando tienes grupos de opciones relacionadas, como varios radio buttons para elegir un plan, usa `<fieldset>` para agruparlos y `<legend>` para ponerle título al grupo. El lector de pantalla anuncia la legend antes de cada opción, así la persona entiende el contexto: "Plan, grupo; opción 1 de 3: Básico". Sin fieldset y legend, las opciones suenan sueltas y confusas.

Hay casos donde no hay un texto visible que sirva como etiqueta, por ejemplo un botón que solo muestra un icono de lupa o una X para cerrar. Para esos casos existe aria-label, un atributo que le pone un nombre accesible invisible: `<button aria-label="Buscar">` con la lupita adentro, o `<button aria-label="Cerrar menú">` con la X. ARIA son las siglas de Aplicaciones de Internet Enriquecidas Accesibles, un conjunto de atributos que le dan información extra a las tecnologías de asistencia. La regla de oro con ARIA es: no uses ARIA si el HTML ya lo dice. Si tienes un texto visible, usa `<label>` o el texto del botón; reserva aria-label para cuando realmente no hay texto visible. El abuso de ARIA es peor que su ausencia, porque ARIA mal usado confunde al lector de pantalla más que el silencio.

Otro atributo ARIA muy útil es aria-describedby, que conecta un campo con un texto de ayuda o de error mediante id. Por ejemplo: `<input id="pass" aria-describedby="ayuda-pass">` junto con `<p id="ayuda-pass">La contraseña debe tener al menos 8 caracteres</p>`. El lector anunciará esa ayuda cuando el usuario llegue al campo. Y cuando el usuario comete un error, no basta con pintar el campo de rojo: debes describir el error con texto, por ejemplo `<p id="error-email">El correo debe contener un @</p>` asociado con aria-describedby, porque el color solo no lo perciben las personas daltónicas ni los lectores de pantalla.

Para cerrar: marca los campos obligatorios con el atributo required y con texto visible que diga "obligatorio", no solo con un asterisco de color. Y nunca desactives el zoom ni uses textos diminutos para "que quepa todo": la accesibilidad también es dejar que cada persona ajuste la página a sus necesidades.

Encabezados con jerarquía: el mapa de tu página

Los encabezados h1 a h6 no son "textos grandes y chicos": son el esqueleto lógico de tu documento, su tabla de contenido. Las personas que ven tu página perciben la estructura de un vistazo por el tamaño y la posición; las personas ciegas hacen lo mismo con el lector de pantalla, que les permite saltar de encabezado en encabezado como quien hojea el índice de un libro. Si tu jerarquía es un desastre, su mapa es un desastre.

Las reglas son simples y estrictas. Primera: un solo h1 por página, que describe el tema completo, normalmente el título principal. Segunda: los h2 marcan las secciones principales, los h3 las subsecciones dentro de cada h2, y así sucesivamente. Tercera: no te saltes niveles; después de un h1 va un h2, no un h3 directo, igual que en un libro no pasas del capítulo al sub-sub-apartado sin pasar por el apartado. Cuarta: el tamaño visual se controla con CSS, nunca elijas un h3 "porque se ve más chico" ni un h1 "porque se ve más grande"; la etiqueta comunica estructura, el CSS comunica apariencia.

Un ejemplo correcto para la página de un curso sería: h1 "Curso completo de HTML5", luego h2 "Unidad 8: Precisión, accesibilidad y buenas prácticas", luego h3 "Entidades y símbolos HTML", h3 "Accesibilidad web", h3 "Buenas prácticas", y dentro de accesibilidad podrías tener h4 si hiciera falta más detalle. El lector de pantalla le presenta esa lista a la persona y ella salta directo a "Buenas prácticas" sin escuchar toda la página. En cambio, si hubieras usado h3 para todo "porque se veían bonitos", el mapa no tendría jerarquía y navegar sería un suplicio.

El error clásico es usar encabezados como atajo de diseño. Lo verás en mucho código de principiantes: un h4 suelto en medio de la página porque "ese tamaño me gustaba". Cada vez que sientas esa tentación, recuerda: usa un `<p>` con una clase de CSS para el tamaño que quieras, y reserva los encabezados para la estructura real. Tu yo del futuro, los lectores de pantalla y Google te lo van a agradecer.

Contraste, foco y skip links

La accesibilidad no termina en el código estructural: también hay decisiones de presentación que se planean desde el HTML. La primera es el contraste entre el texto y el fondo. Un texto gris claro sobre fondo blanco puede verse "elegante y minimalista" para un diseñador joven con vista perfecta, pero para una persona mayor o con baja visión es simplemente ilegible. La guía internacional recomienda una proporción de contraste de al menos 4.5 a 1 para texto normal. No necesitas memorizar fórmulas: existen herramientas gratuitas en línea donde pones tus dos colores y te dicen si pasan o no. El hábito profesional es verificar el contraste de tus combinaciones de colores antes de publicar, igual que verificas la ortografía.

La segunda es el foco visible del teclado. Cuando navegas con la tecla Tab, el navegador marca el elemento activo con un contorno, normalmente un recuadro. Muchos diseñadores lo quitan con CSS porque "se ve feo", dejando a los usuarios de teclado completamente ciegos sobre dónde están parados. La regla es: nunca elimines el indicador de foco sin poner otro en su lugar, por ejemplo un contorno con los colores de tu marca. Y a nivel HTML hay una regla de oro: no uses valores positivos en tabindex como `tabindex="1"` o `tabindex="2"`, porque alteran el orden natural de navegación y crean un caos impredecible. El orden natural del documento, de arriba a abajo según tu HTML, ya es el orden correcto si tu HTML está bien estructurado. Solo en casos avanzados se usa `tabindex="0"` para incluir algo en el orden o `tabindex="-1"` para sacarlo, y eso ya es tema de niveles más avanzados.

La tercera joya es el skip link, o enlace de salto. Imagina una página con un menú de 40 enlaces: un usuario de teclado tendría que pulsar Tab 40 veces en cada página solo para llegar al contenido. El skip link es un enlace que pones como lo primero dentro del `<body>`, algo como `<a href="#contenido">Saltar al contenido principal</a>`, que apunta al id de tu `<main>`. Normalmente está oculto visualmente hasta que recibe el foco del teclado, momento en que aparece. Con una sola pulsación de Tab y Enter, el usuario salta directo al contenido. Es unas pocas líneas de código y una diferencia enorme en la experiencia.

Y un detalle que casi todos olvidan: el atributo lang en la etiqueta `<html>`. Si tu página está en español, escribe `<html lang="es">`. El lector de pantalla usa ese dato para elegir la voz y la pronunciación correctas; sin lang, puede leerte tu página en español con pronunciación de inglés, algo tan incómodo como suena. Es un atributo de dos letras que cambia por completo la experiencia de millones de personas. Ponlo siempre.

Buenas prácticas profesionales: escribe como un pro

Llegamos al tercer pilar de la unidad: los hábitos que distinguen al profesional. Aquí no hay etiquetas nuevas que aprender, sino formas de trabajar. La idea central es esta: el código se lee muchas más veces de las que se escribe. Tú vas a releer tu código cuando algo falle, cuando quieras agregar una función, cuando lo retomes después de meses. Otros programadores lo van a leer cuando trabajen contigo. Los reclutadores lo van a leer cuando evalúen tu portafolio. Escribir HTML limpio no es vanidad, es una inversión que se paga sola una y otra vez.

Piensa en las buenas prácticas como la higiene personal. Nadie te da un premio por lavarte las manos, pero todo funciona mejor cuando lo haces y cuando no lo haces tarde o temprano alguien lo nota, y no de buena manera. Un código con buena higiene se entiende rápido, se modifica sin miedo y se depura en minutos; un código sucio es ese cuarto oscuro donde nadie quiere entrar. Vamos con las siete prácticas esenciales.

1. Indenta tu código

Indentar significa desplazar hacia la derecha las etiquetas que están dentro de otras, para que la jerarquía se vea a simple vista. Cada nivel de anidación lleva típicamente dos espacios más que su padre. Compara mentalmente estas dos versiones: una donde todo el código empieza en la columna cero, una sopa de etiquetas donde no sabes qué está dentro de qué, contra otra donde de un vistazo ves que el `<header>` contiene el `<nav>`, que el `<main>` contiene los `<section>`, y que cada `<li>` vive dentro de su `<ul>`. La versión indentada se lee en segundos; la otra exige un esfuerzo mental agotador.

La indentación tiene además un superpoder práctico: te ayuda a detectar etiquetas sin cerrar. Cuando cada apertura y su cierre están alineados verticalmente, un cierre faltante salta a la vista porque la sangría "se rompe". Es como contar escalones: si falta uno, el ritmo se nota. La mayoría de los editores modernos indentan automáticamente cuando pulsas Tab o Enter; configúralo una vez a dos espacios y olvídate. Y sé consistente: si tu proyecto usa dos espacios, todo el proyecto usa dos espacios. La consistencia es la mitad de la profesionalidad.

2. Usa minúsculas

En HTML5 las mayúsculas y minúsculas técnicamente dan igual para el navegador: `<DIV>` y `<div>` funcionan igual. Pero la convención universal del mundo profesional es escribir todo en minúsculas: etiquetas, atributos y valores. Hay razones históricas, porque en XHTML las mayúsculas eran un error, y razones prácticas: el código en minúsculas se lee más rápido, se mezcla mejor con CSS y JavaScript que también usan minúsculas, y evita errores tontos cuando comparas cadenas. Ver un `<P CLASS="Título">` en código moderno es como ver a alguien con calcetines de diferente color en una junta de trabajo: funciona, pero delata descuido. Acostúmbrate desde hoy: todo en minúsculas, siempre.

3. Comenta con <!-- -->

Los comentarios en HTML se escriben entre <!-- y --> y sirven para dejar notas dentro del código que el navegador ignora pero los humanos leen. Úsalos para marcar las grandes secciones de tu página, como <!-- Encabezado principal --> o <!-- Pie de página -->, para explicar decisiones no obvias, como <!-- Este iframe carga el mapa; tarda 2 segundos -->, y para dejar recordatorios, como <!-- TODO: agregar textos alternativos a las fotos nuevas -->. Un buen comentario responde al "por qué", no al "qué": <!-- Esto es un párrafo --> no aporta nada porque el código ya lo dice, pero <!-- Usamos article porque cada curso se puede syndicar por separado --> explica una decisión que el código por sí solo no revela.

Pero atención a la advertencia seria: los comentarios viajan con tu archivo al navegador de cada visitante, y cualquiera puede verlos con la opción "ver código fuente". Nunca, bajo ninguna circunstancia, escribas en un comentario contraseñas, claves de API, datos personales de usuarios, ni insultos sobre tu jefe o tu cliente. Parece obvio, pero ha pasado en empresas reales y ha salido en las noticias. Los comentarios son para humanos de tu equipo, no para secretos.

4. Usa semántica antes que div

La etiqueta <div> es una caja genérica sin significado: le dice al navegador "aquí hay un bloque" y nada más. Las etiquetas semánticas como <header>, <nav>, <main>, <article>, <section>, <aside> y <footer> dicen qué es cada bloque: "esto es el encabezado", "esto es la navegación", "esto es el contenido principal". Esa información la usan los lectores de pantalla para anunciar roles y permitir saltos rápidos, y la usan los buscadores para entender tu página. Una página hecha solo de divs es como un libro sin capítulos ni títulos: se puede leer, pero navegarlo es un martirio.

La regla práctica es: antes de escribir un <div>, pregúntate si existe una etiqueta semántica que describa mejor ese contenido. ¿Es el encabezado de la página? Usa <header>. ¿Es el menú de navegación? Usa <nav>. ¿Es el contenido principal único de la página? Usa <main>. ¿Es una pieza independiente que tendría sentido por sí sola, como la tarjeta de un curso o una noticia? Usa <article>. ¿Es una agrupación temática dentro de algo más grande? Usa <section>. ¿Es contenido complementario, como una barra lateral? Usa <aside>. ¿Es el cierre con datos de contacto y derechos? Usa <footer>. El <div> queda reservado para cuando de verdad necesitas una caja sin significado, por ejemplo un contenedor solo para aplicar estilos. Semántica primero, div como último recurso.

5. No repitas id

El atributo id es como la CURP de un elemento: un identificador único en todo el documento. Sirve para que CSS, JavaScript, los labels de formularios y las tecnologías de asistencia apunten a un elemento específico sin ambigüedad. Si repites el mismo id en dos elementos, rompes ese contrato: el JavaScript solo encontrará al primero, el label solo se asociará con el primero, y el validador te marcará error. Es un bug silencioso y traicionero porque la página "se ve bien" mientras por dentro las referencias apuntan al lugar equivocado.

La solución es simple: lo que se repite usa class, lo único usa id. Tienes veinte tarjetas de curso iguales: cada tarjeta lleva class="tarjeta-curso", y si una necesita ser referenciada en particular le das un id único como id="curso-destacado". Y cuando asocies labels con inputs mediante for e id, verifica que cada id aparezca una sola vez; un for duplicado es un formulario roto para el lector de pantalla.

6. Válida tu HTML con el validador de la W3C

La W3C, el organismo que define los estándares de la web, ofrece una herramienta gratuita en validator.w3.org donde pegas la dirección de tu página o tu código y te dice con precisión qué está mal: etiquetas sin cerrar, atributos mal escritos, ids duplicados, ampersands sin escapar, jerarquías rotas. Es como un corrector ortográfico pero para HTML, y usarlo debería ser un reflejo automático antes de publicar cualquier cosa.

¿Por qué validar si "mi página se ve bien"? Porque los navegadores son demasiado perdonadores: cuando encuentran un error, cada uno lo "arregla" a su manera con reglas internas de recuperación. El resultado es que tu página con errores se ve bien en tu navegador pero rota en el del cliente, o bien hoy pero mal después de una actualización. El HTML válido se comporta de forma predecible en todos lados. Además, el validador es un maestro paciente: cada error que te marca es una lección gratis sobre algo que estabas haciendo mal. Los profesionales validan; los aficionados confían en que "se ve bien en mi compu".

7. Nombres de archivos sensatos

El nombre de tus archivos también es código, y también tiene buenas prácticas. Usa siempre minúsculas, sin espacios, sin tildes ni ñes, y separa las palabras con guiones: curso-html5.html, no "Curso HTML5 final 2.html". Hay razones técnicas: muchos servidores usan Linux, que distingue entre Mayúsculas y minúsculas, así que Curso.html e curso.html son archivos diferentes y un enlace con la capitalización equivocada se rompe; los espacios en los nombres se convierten en %20 en las URLs, quedando enlaces feos y frágiles; y las tildes y

eñes pueden corromperse al subir archivos a servidores con otra configuración. Además organiza tus archivos en carpetas claras: css/ para estilos, img/ para imágenes, js/ para scripts. Cuando tu proyecto crezca a decenas de archivos, esta disciplina es la diferencia entre encontrar todo en segundos o perder tardes enteras.

Analogías que lo atan todo

Ya conociste varias analogías en esta unidad; vamos a atarlas para que los tres temas queden grabados. Las entidades HTML son códigos secretos, como la clave de los espías en las películas: le dices al navegador en clave "esto es texto, no una orden", y solo quien conoce la clave muestra el mensaje correcto. Sin la clave, el mensaje se malinterpreta y la misión falla. La accesibilidad web son las rampas y los elevadores de los edificios: no son un lujo para pocos, son la garantía de que todos pueden entrar, y cuando se diseñan desde el plano benefician a todo el mundo, desde la silla de ruedas hasta la carriola y el repartidor. Y las buenas prácticas son la higiene: lavarse las manos, tender la cama, ordenar la cocina; nadie te aplaude por hacerlo, pero todo funciona mejor, dura más y cuando alguien lo revisa, se nota la calidad. Códigos secretos para la precisión, rampas para la inclusión, higiene para la profesionalidad: con esas tres imágenes mentales ya tienes la filosofía completa de esta unidad.

Errores comunes que delatan a un principiante

Esta es la lista de los errores que los desarrolladores experimentados detectan en segundos cuando revisan código ajeno. Conócelos para no cometerlos, o mejor, para corregirlos antes de que alguien más los vea.

- Olvidar el punto y coma en las entidades: escribir &lt sin el; hace que la entidad no funcione y se muestre el texto crudo. Toda entidad termina en punto y coma, sin excepciones.
- Usar para indentar o para empujar elementos: cadenas de para "dar espacio" son un fósil de los noventa. Para espacios visuales usa CSS; es solo para mantener dos palabras juntas.
- Abusar de
 para dar espacio vertical: poner

 entre secciones es maquetar con etiquetas de salto de línea. Los espacios entre bloques se controlan con CSS, no con saltos de línea repetidos.

- Usar el placeholder como única etiqueta del campo: el texto desaparece al escribir, muchos lectores no lo anuncian y su contraste suele ser pobre. Siempre un `<label>` real asociado con `for` e `id`.
- Poner `alt="imagen"` o no poner `alt`: el lector ya sabe que es una imagen; necesita saber qué muestra o a dónde lleva. Y las decorativas llevan `alt=""` vacío, nunca omitido.
- Repetir el mismo `id` en varios elementos: rompe JavaScript, confunde labels y es inválido. Lo repetible es `class`, lo único es `id`.
- Escribir etiquetas en mayúsculas: `<DIV>` funciona pero grita "soy nuevo". Todo en minúsculas, siempre.
- Olvidar el `charset` y el `lang`: sin `<meta charset="UTF-8">` arriesgas tildes rotas; sin `<html lang="es">` el lector de pantalla pronuncia mal tu idioma. Dos líneas que cuestan diez segundos.
- Escribir secretos en los comentarios: contraseñas o claves en `<!-- -->` viajan al navegador de todo el mundo. Los comentarios son para notas del equipo, jamás para secretos.
- Saltarse niveles de encabezado: pasar de `h1` a `h3` porque "se veía mejor" rompe el mapa de navegación del lector de pantalla. La jerarquía es lógica, el tamaño visual es trabajo de CSS.

Checklist de calidad: tu lista antes de publicar

Todo piloto usa una lista de verificación antes de despegar, por muy experimentado que sea, porque la memoria falla y los errores cuestan caro. Esta es tu lista de piloto web: recórrela antes de publicar cualquier página, y con el tiempo la harás de memoria.

- ¿Mi documento empieza con `<!DOCTYPE html>` y `<html lang="es">`?
- ¿El `<head>` incluye `<meta charset="UTF-8">` como una de las primeras líneas?
- ¿La página tiene un `<title>` descriptivo y único?
- ¿Hay un solo `<h1>` y los encabezados siguen un orden lógico sin saltos?
- ¿Todas las imágenes tienen `alt` descriptivo, o `alt=""` si son decorativas?
- ¿Cada campo de formulario tiene su `<label>` asociado con `for` e `id`?
- ¿Los botones de solo icono tienen `aria-label` con su nombre?
- ¿Los caracteres `<`, `>` y `&` que van como texto están escapados como entidades?
- ¿El código está indentado de forma consistente con minúsculas en todo?

- ¿Usé etiquetas semánticas (header, nav, main, article, section, footer) en vez de divs genéricos?
- ¿Cada id aparece una sola vez en todo el documento?
- ¿Agregué comentarios en las secciones grandes sin revelar secretos?
- ¿Los nombres de archivos están en minúsculas, sin espacios ni tildes?
- ¿El contraste entre texto y fondo es suficiente para leerse con comodidad?
- ¿Incluí un skip link hacia el contenido principal?
- ¿Pasé la página por el validador de la W3C y corregí los errores?
- ¿Probé navegar la página solo con el teclado, usando Tab y Enter?

Si respondes sí a todo, tu página no solo se ve profesional: lo es. Y cada vez que cumplas esta lista, estarás construyendo la reputación que todo desarrollador quiere: la de alguien en quien se puede confiar.

Ejemplos paso a paso

La teoría ya la tienes; ahora vamos a verla en acción. Cada ejemplo es un archivo completo y funcional que puedes copiar, guardar y abrir en tu navegador para experimentar. Léelos con calma, modifica cosas y observa qué cambia: así se aprende de verdad.

Ejemplo 1: página de tutorial que muestra código escapado con entidades

Este ejemplo es una mini lección que enseña cómo escribir un párrafo en HTML. Fíjate en tres cosas: el `<meta charset="UTF-8">` al inicio del head, la combinación `<pre><code>` para el bloque de código, y cómo cada `<`, `>` y `&` del código de ejemplo está escapado como entidad. Si quitas los escapes, el tutorial se rompe: el navegador ejecutaría el código en vez de mostrarlo.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Mini tutorial: tu primer párrafo</title>
</head>
<body>
  <!-- Encabezado de la lección -->
  <header>
    <h1>Cómo escribir un párrafo en HTML</h1>
    <p>En esta lección aprenderás la etiqueta más usada de todas: el párrafo.</p>
```

```

</header>

<!-- Contenido principal de la lección -->
<main>
  <h2>El código</h2>
  <p>Para crear un párrafo, escribe tu texto entre las etiquetas de apertura y
  cierre:</p>

  <!-- Bloque de código de ejemplo: fíjate que <, > están escapados
  como &lt; y &gt; para que se muestren como texto y no se ejecuten.
  El <pre> conserva los saltos de línea y el <code> indica que es código. -->
  <pre><code>&lt;p&gt;Hola mundo, este es mi primer párrafo.&lt;/p&gt;</code></pre>

  <h2>El resultado</h2>
  <p>El navegador muestra el siguiente párrafo real:</p>
  <p>Hola mundo, este es mi primer párrafo.</p>

  <h2>Un ejemplo con ampersand</h2>
  <p>Si tu código de ejemplo incluye un &, también debes escaparlo como
  &amp;</p>
  <pre><code>&lt;p&gt;Términos &amp; condiciones&lt;/p&gt;</code></pre>
  <p>El visitante verá en pantalla: &lt;p&gt;Términos &amp; condiciones&lt;/p&gt;
  como texto.</p>
</main>

<!-- Pie con símbolo de derechos de autor usando entidad -->
<footer>
  <p>&copy; 2026 Lucas EGA Academy. Todos los derechos reservados.</p>
</footer>
</body>
</html>

```

Prueba este experimento: cambia `<p>` por `<p>` dentro del `<pre><code>` y recarga. Verás cómo el ejemplo desaparece y en su lugar aparece un párrafo real. Ese es exactamente el problema que las entidades resuelven.

Ejemplo 2: formulario de registro accesible con labels y ARIA

Este formulario aplica todo lo aprendido sobre accesibilidad: cada campo tiene su `<label>` asociado con `for` e `id`, los radio buttons van agrupados en `<fieldset>` con `<legend>`, los campos obligatorios usan `required` más texto visible, la ayuda de la contraseña se conecta con `aria-describedby`, y el botón de solo icono lleva `aria-label`. Además incluye un skip link al inicio del body.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Registro - Academia de ejemplo</title>
</head>
<body>
  <!-- Skip link: lo primero en el body, salta al contenido principal -->
  <a href="#contenido">Saltar al contenido principal</a>

  <header>
    <h1>Crea tu cuenta gratis</h1>
    <!-- Botón solo con icono: aria-label le da nombre accesible -->
    <button type="button" aria-label="Abrir menú de navegación">X</button>
  </header>

  <main id="contenido">
    <h2>Formulario de registro</h2>
    <p>Los campos marcados con (obligatorio) no pueden quedar vacíos.</p>

    <form action="/registro" method="post">
      <!-- Cada label está asociado a su input con for + id iguales -->
      <div>
        <label for="nombre">Nombre completo (obligatorio)</label>
        <input type="text" id="nombre" name="nombre" required placeholder="Ej. María
García">
      </div>

      <div>
        <label for="email">Correo electrónico (obligatorio)</label>
        <input type="email" id="email" name="email" required placeholder="Ej.
maria@correo.com">
        <!-- Mensaje de error con texto, no solo color, conectado con aria-describedby
-->
        <p id="error-email">El correo debe contener un @ y un dominio válido.</p>
      </div>

      <div>
        <label for="pass">Contraseña (obligatorio)</label>
        <!-- aria-describedby conecta el campo con su texto de ayuda -->
        <input type="password" id="pass" name="pass" required
aria-describedby="ayuda-pass">
        <p id="ayuda-pass">Mínimo 8 caracteres, con al menos un número.</p>
      </div>
    </form>
  </main>

```

```

<!-- Grupo de opciones relacionadas: fieldset + legend -->
<fieldset>
  <legend>¿Cómo te enteraste de nosotros? (obligatorio)</legend>
  <div>
    <input type="radio" id="plan-amigo" name="origen" value="amigo" required>
    <label for="plan-amigo">Me lo recomendó un amigo</label>
  </div>
  <div>
    <input type="radio" id="plan-redes" name="origen" value="redes">
    <label for="plan-redes">Lo vi en redes sociales</label>
  </div>
  <div>
    <input type="radio" id="plan-buscador" name="origen" value="buscador">
    <label for="plan-buscador">Lo encontré en el buscador</label>
  </div>
</fieldset>

<div>
  <input type="checkbox" id="noticias" name="noticias">
  <label for="noticias">Quiero recibir noticias y cursos nuevos</label>
</div>

  <button type="submit">Crear mi cuenta</button>
</form>
</main>

<footer>
  <p>&copy; 2026 Academia de ejemplo.</p>
</footer>
</body>
</html>

```

Navega este formulario solo con el teclado: pulsa Tab para avanzar y verás cómo el foco se mueve en orden lógico, cómo cada campo anuncia su etiqueta y cómo el primer Tab te ofrece saltar al contenido. Eso es accesibilidad funcionando.

Ejemplo 3: página antes y después aplicando buenas prácticas

Primero la versión "antes": funciona y se ve igual, pero es un desastre por dentro: sin indentar, mayúsculas mezcladas, divs para todo, sin alt, sin lang ni charset, id repetido y entidades sin escapar.

```
<!-- VERSIÓN ANTES: no copies estos hábitos -->
<HTML><HEAD><TITLE>curso</TITLE></HEAD>
<BODY>
<div id="tarjeta"><div id="tarjeta">
<P>Curso de HTML5</P><IMG SRC="curso.jpg"><BR><BR>
<A HREF="#">Ver curso</A></div></div>
<!-- precio: $299, no decirle a nadie -->
</BODY></HTML>
```

Ahora la versión "después": el mismo contenido, pero con HTML profesional. Compáralas línea por línea y verás cada buena práctica aplicada: doctype, lang, charset, título descriptivo, minúsculas, indentación, semántica con article, alt descriptivo, ids únicos, espaciado con estructura en vez de

, y comentarios útiles.

```

<!-- VERSIÓN DESPUÉS: así escribe un profesional -->
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Curso de HTML5 - Lucas EGA Academy</title>
</head>
<body>
  <main>
    <!-- Tarjeta de curso: article porque es contenido independiente -->
    <article class="tarjeta-curso" id="curso-html5">
      <h2>Curso de HTML5</h2>
      
      <p>Aprende HTML5 desde cero con 30 clases prácticas y ejemplos reales.</p>
      <p>Precio: $299</p>
      <a href="#">Ver curso</a>
    </article>

    <article class="tarjeta-curso" id="curso-css3">
      <h2>Curso de CSS3</h2>
      
      <p>Dale estilo profesional a tus páginas con CSS moderno.</p>
      <p>Precio: $299</p>
      <a href="#">Ver curso</a>
    </article>
  </main>
</body>
</html>

```

Fíjate en los detalles: las dos tarjetas usan `class="tarjeta-curso"` porque se repiten, pero cada una tiene un `id` único; el precio ya no está escondido en un comentario conspiranoico; y no hay ni un `<br>` usado como espaciado. El navegador muestra lo mismo, pero este código se puede mantener, validar y entender.

Ejemplo 4: documento completo bien indentado y comentado

Para cerrar, una página completa de artículo de blog que reúne todo: estructura semántica, indentación consistente de dos espacios, comentarios que marcan secciones, entidades donde corresponden, `alt` descriptivos, jerarquía de encabezados correcta y `skip link`. Úsala como plantilla para tus propios proyectos.


```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Qué son las entidades HTML - Blog de Lucas EGA Academy</title>
</head>
<body>
  <!-- Enlace de salto para usuarios de teclado -->
  <a href="#contenido">Saltar al contenido principal</a>

  <!-- Encabezado del sitio con navegación principal -->
  <header>
    <p><a href="/">Lucas EGA Academy</a></p>
    <nav aria-label="Navegación principal">
      <ul>
        <li><a href="/cursos">Cursos</a></li>
        <li><a href="/blog">Blog</a></li>
        <li><a href="/contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>

  <!-- Contenido principal: un solo main por página -->
  <main id="contenido">
    <!-- Artículo independiente: tiene sentido por sí solo -->
    <article>
      <header>
        <h1>Qué son las entidades HTML y cuándo usarlas</h1>
        <p>Publicado el 19 de septiembre de 2026 | 8 minutos de lectura</p>
      </header>

       y & escritas con tiza">

      <h2>El problema de los caracteres reservados</h2>
      <p>El navegador interpreta los signos <, > y & como instrucciones. Para mostrarlos como texto usamos entidades.</p>

      <h2>Las más usadas</h2>
      <ul>
        <li>&nbsp; para el espacio inseparable</li>
        <li>&lt; y &gt; para mostrar etiquetas</li>
        <li>&amp; para el ampersand en texto normal</li>
        <li>&copy; para el símbolo ©</li>

```

```

    </ul>

    <footer>
      <p>Escrito por el equipo de Lucas EGA Academy.</p>
    </footer>
  </article>

  <!-- Contenido complementario: barra lateral -->
  <aside>
    <h2>Artículos relacionados</h2>
    <ul>
      <li><a href="#">Valida tu HTML con la W3C</a></li>
      <li><a href="#">Accesibilidad en 10 pasos</a></li>
    </ul>
  </aside>
</main>

<!-- Pie de página del sitio -->
<footer>
  <p>&copy; 2026 Lucas EGA Academy. Todos los derechos reservados.</p>
</footer>
</body>
</html>

```

Observa cómo los comentarios marcan cada sección grande sin explicar lo obvio, cómo la indentación revela la jerarquía de un vistazo, y cómo dentro del artículo las entidades aparecen doblemente escapadas (&lt;) porque ahí estamos mostrando el nombre de la entidad como texto. Este es el nivel de acabado al que debes aspirar en cada página que publiques.

Práctica

Es hora de ensuciarte las manos. Cada ejercicio trae su solución explicada paso a paso, pero te recomiendo intentarlo primero por tu cuenta antes de leerla: equivocarte y corregir es donde ocurre el verdadero aprendizaje.

Ejercicio 1: escapa un fragmento de código para mostrarlo como texto

Enunciado: Tienes un blog donde quieres publicar el siguiente fragmento de HTML como ejemplo visible para tus lectores. Escríbelo escapado con entidades dentro de la estructura `<pre><code>`, listo para pegarlo en tu página. El fragmento es:

`<p class="aviso">Ofertas & descuentos <strong>hoy</strong></p>`

Solución paso a paso:

Paso 1: Identifica los caracteres reservados en el fragmento. Hay signos de menor que y mayor que en `<p>`, `</p>`, `<strong>` y `</strong>`. Además hay un ampersand en "Ofertas & descuentos". En total debes escapar `<`, `>` y `&`.

Paso 2: Escapa primero el ampersand que forma parte del texto, para no interferir con las entidades que vas a crear después. "Ofertas & descuentos" se convierte en "Ofertas & descuentos".

Paso 3: Escapa los signos de menor que y mayor que. Cada `<` se vuelve `<` y cada `>` se vuelve `>`. El fragmento queda así: `<p class="aviso">Ofertas & descuentos <strong>hoy</strong></p>`. Fíjate que las comillas del atributo `class` no necesitan escaparse porque están dentro del texto del ejemplo y no rompen nada.

Paso 4: Envuelve el resultado en `<pre><code>` para conservar el formato e indicar que es código. El bloque final listo para tu página es:

```
<pre><code>&lt;p          class="aviso"&gt;Ofertas          &amp;          descuentos
&lt;strong&gt;hoy&lt;/strong&gt;&lt;/p&gt;</code></pre>
```

Paso 5: Verifica abriendo la página en el navegador. Debes ver exactamente el fragmento original como texto: `<p class="aviso">Ofertas & descuentos <strong>hoy</strong></p>`, sin que se ejecute. Si ves `<` literal en pantalla, escapaste dos veces; si ves el párrafo ejecutado, te faltó escapar.

Ejercicio 2: corrige los textos alternativos de tres imágenes

Enunciado: Una página de cursos tiene estas tres imágenes con alt incorrectos. Escribe un alt correcto para cada una y explica qué estaba mal:

1. `` (el logo, que es un enlace al inicio)
2. `` (foto del profesor en el pizarrón)
3. `` (una línea decorativa entre secciones)

Solución paso a paso:

Paso 1: Analiza la imagen 1. El `alt="imagen"` es redundante: el lector de pantalla ya anuncia que es una imagen. Además el logo es un enlace al inicio, así que el alt debe decir a dónde

lleva, no qué muestra. Alt correcto: alt="Lucas EGA Academy - inicio". Así la persona ciega escucha "enlace, Lucas EGA Academy - inicio" y sabe exactamente qué hace ese elemento.

Paso 2: Analiza la imagen 2. El alt="foto-profe.jpg" anuncia el nombre del archivo, que no le dice nada útil a nadie. El alt debe describir el contenido relevante: alt="Profesor explicando la estructura de una página HTML frente a un pizarrón verde". Una frase, contenido esencial, sin empezar con "imagen de".

Paso 3: Analiza la imagen 3. Es decorativa: no aporta información, solo adorna. El alt="adorno bonito" obliga al lector a escuchar ruido irrelevante. Alt correcto: alt="" vacío, para que el lector de pantalla la ignore por completo.

Paso 4: Regla general que acabas de aplicar: toda img lleva alt; si informa, describe su contenido o función en una frase; si es enlace, describe el destino o la acción; si decora, alt vacío. Repasa tus propios proyectos y corrige cada alt con este criterio.

Ejercicio 3: haz accesible un formulario roto

Enunciado: Este formulario de contacto tiene cuatro problemas de accesibilidad. Encuéntralos y escribe la versión corregida:

```
<form action="/contacto" method="post">
  <input type="text" id="c1" placeholder="Tu nombre">
  <input type="email" id="c2" placeholder="tu@correo.com">
  <button>X</button>
</form>
```

Solución paso a paso:

Paso 1: Detecta el problema uno: no hay <label> en ningún campo, solo placeholders que desaparecen al escribir y que muchos lectores no anuncian. Cada input necesita un label visible asociado.

Paso 2: Detecta el problema dos: el botón solo dice "X", que para un lector de pantalla suena como "botón equis" sin explicar qué hace. Necesita un aria-label que le dé nombre, por ejemplo aria-label="Enviar mensaje".

Paso 3: Detecta el problema tres: los ids genéricos c1 y c2 no son descriptivos y dificultan asociar y mantener el código. Usa ids con significado: nombre y email.

Paso 4: Detecta el problema cuatro: no se indica cuáles campos son obligatorios ni hay texto de ayuda. Agrega required y mención visible de obligatoriedad.

Paso 5: Escribe la versión corregida aplicando todo:

```
<form action="/contacto" method="post">
  <div>
    <label for="nombre">Nombre (obligatorio)</label>
    <input type="text" id="nombre" name="nombre" required placeholder="Ej. María
García">
  </div>
  <div>
    <label for="email">Correo electrónico (obligatorio)</label>
    <input type="email" id="email" name="email" required placeholder="Ej.
maria@correo.com">
  </div>
  <button type="submit" aria-label="Enviar mensaje">X</button>
</form>
```

Paso 6: Comprueba con el teclado: Tab debe moverse en orden, cada campo debe anunciar su etiqueta, y el botón debe anunciarse como "Enviar mensaje". Si algo no se anuncia, revisa que cada for coincida exactamente con su id.

Ejercicio 4: reescribe una página con semántica y buenas prácticas

Enunciado: Reescribe este fragmento aplicando todo lo aprendido: minúsculas, indentación, etiquetas semánticas en vez de div, alt descriptivo, un solo h1 con jerarquía correcta e ids únicos.

```
<DIV id="caja"><DIV id="caja">
<H3>Bienvenidos a mi pagina</H3>
<IMG src="banner.jpg">
<P>Aprende programacion desde cero.</P>
</DIV></DIV>
```

Solución paso a paso:

Paso 1: Convierte todo a minúsculas y aplica indentación de dos espacios por nivel. Ya con eso el código respira.

Paso 2: Sustituye los divs genéricos por semántica: el contenedor exterior puede ser un <main> si es el contenido principal, o un <section>; cada tarjeta o bloque independiente merece un <article>. Aquí usaremos <main> con un <article> dentro.

Paso 3: Corrige la jerarquía de encabezados: ese h3 suelto debe ser h1 si es el título principal de la página, o h2 si vive dentro de una página que ya tiene h1. Lo convertimos en h1 por ser el título principal.

Paso 4: Agrega el alt descriptivo a la imagen: alt="Estudiante programando en una laptop con código en la pantalla".

Paso 5: Cambia los ids duplicados id="caja" por clases para lo repetible y un id único y descriptivo donde haga falta, por ejemplo class="contenedor" e id="bienvenida".

Paso 6: El resultado final:

```
<main class="contenedor" id="bienvenida">
  <article>
    <h1>Bienvenidos a mi página</h1>
    
    <p>Aprende programación desde cero.</p>
  </article>
</main>
```

Paso 7: Pásalo por el validador de la W3C para confirmar que no quedaron errores. Compara el antes y el después: el navegador muestra lo mismo, pero el después es accesible, mantenible y profesional.

Ejercicio 5: audita una página con el checklist de calidad

Enunciado: Un compañero te pide revisar su página antes de publicarla. Aplica el checklist de la unidad y encuentra al menos ocho fallos en este código. Para cada fallo indica qué está mal, por qué importa y cómo corregirlo.

```
<HTML>
<HEAD>
<TITLE>Mi pagina</TITLE>
</HEAD>
<BODY>

<h2>Ofertas de hoy</h2>
<h4>Curso de HTML</h4>
<p>Aprende & diviértete</p>
<form>
<input type="text" id="dato" placeholder="Nombre">
<input type="text" id="dato" placeholder="Ciudad">
</form>
<div><div><div>© 2026 Mi Sitio</div></div></div>
</BODY>
</HTML>
```

Solución paso a paso:

Fallo 1: Falta `<!DOCTYPE html>` al inicio. Sin doctype el navegador entra en "modo raro" y cada navegador interpreta el diseño a su manera. Corrección: poner `<!DOCTYPE html>` como primera línea.

Fallo 2: La etiqueta `html` no tiene `lang`. Sin `<html lang="es">` el lector de pantalla puede pronunciar el español con voz de otro idioma. Corrección: `<html lang="es">`.

Fallo 3: Falta `<meta charset="UTF-8">` en el head. Las tildes de "página" podrían romperse en algunos navegadores. Corrección: agregarla como primera etiqueta del head.

Fallo 4: Todo está en mayúsculas (`<HTML>`, `<HEAD>`, `<TITLE>`). Funciona pero es mala práctica y delata descuido. Corrección: todo en minúsculas.

Fallo 5: La imagen no tiene `alt`. El lector anunciará "foto punto jpg" o nada útil. Corrección: agregar `alt` descriptivo, por ejemplo `alt="Estudiantes en clase de programación"`, o `alt=""` si fuera decorativa.

Fallo 6: La jerarquía de encabezados está rota: empieza en `h2` y salta a `h4` sin `h1` ni `h3`. El mapa de navegación del lector queda sin sentido. Corrección: un `h1` para el título principal, luego `h2` para secciones, sin saltos.

Fallo 7: El ampersand en "Aprende & diviértete" no está escapado. Debe ser "Aprende & diviértete" para que el HTML sea válido.

Fallo 8: Los dos inputs repiten `id="dato"` y no tienen `<label>`. El id duplicado es inválido y rompe la asociación; sin label el lector no sabe qué pedir en cada campo. Corrección: ids únicos (nombre, ciudad) y un `<label>` asociado a cada uno con `for`.

Fallo 9: El pie de página usa tres divs anidados para un simple texto, pudiendo ser un `<footer>` con un `<p>`, y el © debería escribirse como `©` por precisión. Corrección: `<footer><p>© 2026 Mi Sitio</p></footer>`.

Fallo 10: El código no está indentado, lo que dificulta detectar la estructura y los errores. Corrección: indentar dos espacios por nivel.

Conclusión del ejercicio: diez fallos en veinte líneas, y la página "se veía bien". Esa es exactamente la lección de esta unidad: lo que se ve bien por fuera puede estar roto por dentro, y tu trabajo como profesional es que esté bien por dentro también. Usa este ejercicio como modelo cada vez que revises código ajeno o el tuyo propio.

Resumen de la unidad

- Las entidades HTML son códigos que empiezan con `&` y terminan con `;` y le dicen al navegador que muestre un carácter como texto en vez de interpretarlo como instrucción.
- Los tres caracteres reservados que siempre debes escapar como texto son `<` (`<`), `>` (`>`) y `&` (`&`); el ampersand es el más traicionero porque inicia las entidades.
- ` ` es el espacio inseparable: mantiene dos palabras juntas en la misma línea, como en `10 km`; no lo uses para indentar ni para dar espacios visuales, para eso existe CSS.
- Las entidades imprescindibles son `<`, `>`, `&`, `"`, `©`, `®`, `™`, `€`, `£`, `¥` y `¢`; con ellas cubres casi todo lo que necesitarás.
- Para mostrar código como texto sin que se ejecute, escapa `<`, `>` y `&` y envuelve el resultado en `<pre><code>`; el `<pre>` conserva el formato y el `<code>` indica que es código.
- Todo documento HTML debe llevar `<meta charset="UTF-8">` como una de las primeras etiquetas del `<head>` para que tildes, eñes y símbolos se muestren correctamente en cualquier navegador.
- Con UTF-8 declarado puedes escribir tildes y eñes directamente; las entidades quedan reservadas para los caracteres reservados y para símbolos especiales cuando te convenga.

- La accesibilidad web significa que cualquier persona pueda usar tu sitio, incluyendo quienes navegan con lector de pantalla, solo con teclado o con baja visión; es un deber ético, muchas veces legal, y además amplía tu audiencia y mejora tu SEO.
- Toda imagen lleva alt: descriptivo en una frase si la imagen informa, con la acción o destino si es enlace o botón, y vacío (alt="") si es puramente decorativa.
- Cada campo de formulario necesita un <label> visible asociado con for e id; el placeholder es solo una ayuda opcional y nunca sustituye al label.
- Usa <fieldset> con <legend> para agrupar opciones relacionadas, aria-label para nombrar botones sin texto visible y aria-describedby para conectar ayudas y mensajes de error.
- No uses ARIA cuando el HTML semántico ya expresa lo mismo; el abuso de ARIA confunde más que su ausencia.
- Usa un solo h1 por página y una jerarquía de encabezados sin saltos de nivel; los encabezados son el mapa de navegación del lector de pantalla, no un atajo de diseño.
- Verifica el contraste entre texto y fondo, nunca elimines el foco visible del teclado sin reemplazo y pon un skip link al inicio del body que salte al contenido principal.
- Declara siempre <html lang="es"> para que el lector de pantalla pronuncie tu página en el idioma correcto.
- Indenta tu código con dos espacios por nivel, escribe todo en minúsculas y comenta las secciones grandes con <!-- --> explicando el porqué, nunca lo obvio y jamás secretos.
- Prefiere etiquetas semánticas (header, nav, main, article, section, aside, footer) antes que div, y reserva el div para cajas sin significado real.
- El id es único en todo el documento como una CURP; lo que se repite usa class y lo único usa id.
- Válida tu HTML en validator.w3.org antes de publicar: los navegadores perdonan errores de formas distintas y el HTML válido se comporta igual en todos lados.
- Nombra tus archivos en minúsculas, sin espacios, sin tildes ni eñes, con guiones para separar palabras, y organízalos en carpetas claras como css/, img/ y js/.
- Recuerda las tres analogías: las entidades son códigos secretos, la accesibilidad son rampas para que todos entren y las buenas prácticas son higiene que mantiene sano tu código.

- Antes de publicar cualquier página, recorre el checklist de calidad de esta unidad: doctype, lang, charset, título, jerarquía, alt, labels, entidades, indentación, semántica, ids únicos, contraste, skip link, validación y prueba con teclado.
- A partir de hoy tu HTML no solo funciona: es preciso, es para todos y está escrito como el de un profesional. Esa es la diferencia que los clientes, los equipos y los reclutadores notan aunque no sepan explicar por qué.

UNIDAD 9: Formularios profesionales y estructura del proyecto

Llegaste a la unidad 9, y esta unidad es diferente a todas las anteriores. Hasta aquí aprendiste las piezas del lenguaje una por una: encabezados, párrafos, listas, enlaces, imágenes, tablas, formularios básicos y etiquetas semánticas. Fue como aprender las piezas de un juego de construcción por separado. Pues bien, a partir de esta unidad el curso cambia de marcha: ya no se trata solo de aprender piezas nuevas, se trata de usar todo lo que sabes para construir algo real, algo tuyo, algo que puedas mostrar con orgullo. Esta unidad marca el inicio de tu proyecto final: tu portafolio web personal, la página que te va a presentar al mundo como desarrollador o desarrolladora web.

Esta unidad cubre tres temas grandes, y cada uno tiene una razón de peso para estar aquí. Primero, los formularios profesionales: vas a dominar `fieldset` y `legend` para agrupar campos, `datalist` para dar sugerencias inteligentes, `optgroup` para organizar opciones dentro de un `select`, el `select multiple` y el control fino de `textarea` con filas y columnas. Si la unidad 8 te enseñó a hacer formularios que funcionan, esta unidad te enseña a hacer formularios que se ven y se sienten profesionales, como los de las páginas que usas todos los días. Segundo, las etiquetas obsoletas: `center`, `font`, `marquee` y la diferencia entre `b` e `i` contra `strong` y `em`. Internet está lleno de tutoriales viejos y código heredado que todavía usan estas etiquetas, y si no sabes que están muertas, las vas a copiar sin darte cuenta y vas a construir sobre cimientos podridos. Tercero, el proyecto: la estructura base de tu portafolio, con `header`, `nav`, `main` y `footer` bien armados, y una organización de archivos y carpetas como la de un profesional.

Por que importa tanto esta combinación. Piensalo así: un formulario es el punto donde tu página deja de ser un folleto y se convierte en una conversación. Sin formularios, el usuario solo lee. Con formularios, el usuario responde, se registra, pregunta, compra, opina. Las empresas viven de sus formularios: registros, inicios de sesión, pagos, encuestas, contactos. Un formulario mal organizado espanta usuarios; un formulario claro y profesional genera confianza y conversiones. Cuando domines `fieldset`, `legend`, `datalist` y `optgroup`, tus formularios van a transmitir algo que pocos principiantes transmiten: orden. Y el orden, en la web, se traduce en credibilidad.

Las etiquetas obsoletas importan por una razón distinta pero igual de seria: te protegen del pasado. La web tiene más de treinta años, y en esos años HTML cambió muchísimo. Etiquetas que en 1998 eran la única forma de centrar un texto o ponerle color, hoy están oficialmente

retiradas del estandar. El problema es que miles de tutoriales, videos y foros viejos siguen recomendandolas, y un principiante que no sabe la historia las aprende como si fueran validas. Esta unidad te vacuna contra eso: vas a entender por que murieron, que usar en su lugar y como reconocer código viejo a kilometros de distancia. Un desarrollador que escribe `<font>` en 2026 se delata solo; tu no vas a ser ese desarrollador.

Y el proyecto importa porque es donde todo cobra sentido. Aprender etiquetas sueltas sin un proyecto es como aprender palabras sueltas de un idioma sin nunca tener una conversación: sabes mucho y no puedes decir nada. Tu portafolio web personal va a ser la prueba viviente de lo que sabes. Lo vas a construir poco a poco en las unidades que faltan, y al final del curso tendras una página real, publicada en tu mente como producto terminado, que muestre quien eres, que sabes hacer y como contactarte. Esta unidad pone los cimientos y levanta el esqueleto: la estructura base sobre la que todo lo demas se va a sostener. Si la base queda torcida, todo lo que construyas encima saldra torcido; si queda sólida, cada unidad siguiente sera simplemente agregar pisos a un edificio bien cimentado.

Asi encaja esta unidad en el curso completo. Las unidades 1 a 4 te dieron el esqueleto del documento y el texto: estructura básica, encabezados, párrafos, formato, listas y enlaces. Las unidades 5 y 6 te dieron los sentidos: imágenes, audio, video y tablas para presentar datos. Las unidades 7 y 8 te dieron interactividad y significado: contenido embebido, formularios basicos y HTML semantico. Ahora la unidad 9 toma todo eso y lo sube de nivel: formularios de nivel profesional, criterio para distinguir el HTML bueno del obsoleto, y el inicio del proyecto que va a unificar todo tu aprendizaje. Las unidades que vienen después seguiran alimentando el portafolio: más secciones, mejores formularios de contacto, detalles que lo hagan brillar. Pero el momento en que tu aprendizaje se vuelve un producto real empieza aquí, en esta página.

Al terminar esta unidad vas a ser capaz de tres cosas concretas. Una: construir un formulario de registro completo y profesional, con campos agrupados por temas, sugerencias automaticas y opciones organizadas, que se vea como hecho por alguien que sabe. Dos: mirar cualquier código HTML y detectar al instante las etiquetas obsoletas, explicando por que no deben usarse y con que reemplazarlas. Tres: tener en tu computadora la estructura base de tu portafolio: una carpeta de proyecto organizada como la de un profesional, con un `index.html` que tenga header con logo y navegación, main con contenido principal y footer con derechos reservados. No son promesas vagas: son entregables que vas a poder abrir en tu navegador y mostrarle a quien quieras.

Un consejo antes de empezar: estudia esta unidad con tu editor abierto y el navegador al lado. Cada vez que veas un ejemplo, copialo, pegalo, modificalo y rompe algo a proposito para ver

que pasa. La teoría de esta unidad es importante, pero el músculo se construye practicando. Y con el proyecto, tomate en serio la planificación: un portafolio bien planeado en papel se construye en la mitad de tiempo que uno improvisado. Tu yo del futuro, el que este terminando el curso con un portafolio completo, te va a agradecer que hayas empezado con orden desde la unidad 9.

Explicación detallada

Formularios profesionales: fieldset y legend, tus organizadores de confianza

Imagina que entras a una oficina y encuentras todos los papeles revueltos en una sola pila gigante: facturas mezcladas con contratos, recibos entre cartas personales. Encontrar algo ahí es una pesadilla. Ahora imagina la misma oficina con un archivero: una carpeta para facturas, otra para contratos, otra para recibos, cada una con su etiqueta al frente. Esa es exactamente la diferencia entre un formulario sin organización y un formulario que usa fieldset y legend. El fieldset es la carpeta del archivero: agrupa campos relacionados dentro de un recuadro visible. El legend es la etiqueta pegada al frente de la carpeta: le pone título al grupo para que el usuario sepa que va a encontrar adentro.

La sintaxis es sencilla. Abres `<fieldset>`, pones `<legend>` como primer hijo con el título del grupo, luego metes los campos del grupo, y cierras `</fieldset>`. El navegador dibuja automáticamente un borde alrededor del grupo con el título incrustado en la línea superior del borde, un efecto visual clásico que los usuarios reconocen al instante como "estos campos van juntos". Mira la estructura básica:

```

<form action="/registro" method="post">
  <fieldset>
    <legend>Datos personales</legend>
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required>
    <label for="email">Correo:</label>
    <input type="email" id="email" name="email" required>
  </fieldset>
  <fieldset>
    <legend>Datos de la cuenta</legend>
    <label for="usuario">Usuario:</label>
    <input type="text" id="usuario" name="usuario" required>
    <label for="clave">Contraseña:</label>
    <input type="password" id="clave" name="clave" required>
  </fieldset>
  <button type="submit">Registrarse</button>
</form>

```

Fíjate en algo importante: el legend debe ser el primer hijo del fieldset. Si pones un input antes del legend, el HTML sigue funcionando pero el resultado visual se rompe y los lectores de pantalla para personas con discapacidad visual pueden confundirse. Y hablando de accesibilidad, fieldset y legend no son solo decoración: los lectores de pantalla anuncian el legend antes de cada campo del grupo, así que una persona que no ve la pantalla escucha "Datos personales, Nombre" en lugar de solo "Nombre" suelto. Eso es diseño inclusivo con dos etiquetas.

El fieldset tiene un atributo muy útil que pocos principiantes conocen: disabled. Si pones `<fieldset disabled>`, todos los campos dentro del grupo se desactivan de golpe, sin tener que marcar uno por uno. Es perfecto para formularios por pasos o para secciones que dependen de una condición. Por ejemplo, un grupo de "Datos de facturación" que solo se activa si el usuario marca la casilla "Quiero factura". También existe el atributo form, que permite asociar un fieldset con un formulario aunque este físicamente fuera de él, usando el id del formulario. No lo vas a necesitar todos los días, pero cuando lo necesites te va a salvar.

Errores comunes con fieldset y legend. El primero: usar fieldset para cada campo individual, creando un bosque de recuadros que confunde más de lo que ayuda. El fieldset agrupa campos relacionados, no campos solitarios; si un grupo tiene un solo campo, probablemente no necesita fieldset. El segundo: olvidar el legend. Un fieldset sin legend es como una carpeta del archivero sin etiqueta: el recuadro aparece pero nadie sabe que significa. El tercero: anidar fieldsets sin necesidad, grupos dentro de grupos dentro de grupos, que vuelven el formulario

un laberinto. Un nivel de agrupación basta en el 95 por ciento de los casos; dos niveles solo en formularios realmente largos como solicitudes de empleo o tramites.

Buenas practicas: agrupa por temas logicos que el usuario reconozca (datos personales, dirección, pago), usa legends cortos y claros (dos o tres palabras), y mantén el mismo estilo visual en todos los grupos. Si un día aprendes CSS, podras estilizar el borde del fieldset y el legend a tu gusto; por ahora, el estilo por defecto del navegador ya comunica orden por si solo.

datalist: sugerencias inteligentes sin complicarte la vida

Hay un dilema clásico en los formularios: a veces quieres guiar al usuario con opciones predefinidas, pero no quieres obligarlo a elegir solo entre esas opciones. Un `<select>` lo obliga: solo puede elegir lo que está en la lista. Un `<input>` de texto libre lo deja solo: tiene que adivinar que escribir. El `<datalist>` resuelve el dilema con elegancia: es una lista de sugerencias atada a un campo de texto. El usuario puede elegir una sugerencia con un clic, o ignorarla y escribir lo que quiera. Es lo mejor de los dos mundos.

Funciona con dos piezas conectadas por un id. Primero el input normal, con el atributo `list` apuntando a un identificador. Luego el `datalist` con ese mismo id, que contiene elementos `<option>` con las sugerencias. Mira:

```
<label for="ciudad">Ciudad:</label>
<input type="text" id="ciudad" name="ciudad" list="ciudades">
<datalist id="ciudades">
  <option value="Ciudad de Mexico">
  <option value="Guadalajara">
  <option value="Monterrey">
  <option value="Puebla">
  <option value="Tijuana">
</datalist>
```

Cuando el usuario hace clic en el campo o empieza a escribir, el navegador muestra las sugerencias y las filtra mientras escribe. Si escribe "Gua", solo queda Guadalajara. Si su ciudad no está en la lista, la escribe libremente y el formulario la acepta sin protestar. Fijate que los `option` del `datalist` no llevan etiqueta de cierre en este estilo clásico (aunque ponerla no hace dano); lo importante es el atributo `value`, que es lo que se sugiere.

Cuando conviene `datalist` y cuando conviene `select`. Usa `datalist` cuando la lista de opciones validas es larga o abierta: ciudades, profesiones, lenguajes de programación, marcas de autos. El usuario agradece no tener que recorrer una lista de doscientas ciudades. Usa `select`

cuando las opciones son cerradas y limitadas: día de la semana, metodo de pago, talla de playera. Una regla práctica: si la lista tiene más de quince opciones o el usuario podría querer escribir algo fuera de la lista, datalist es tu amigo.

Errores comunes: el más frecuente es que el valor del atributo list no coincida exactamente con el id del datalist, incluyendo mayúsculas y minúsculas. Si el input dice list="ciudades" y el datalist tiene id="Ciudades", no se conectan y las sugerencias nunca aparecen, sin ningún mensaje de error que te avise. HTML no te regaña: simplemente no funciona y te deja adivinando. El segundo error es poner el datalist sin input asociado, esperando que funcione solo: el datalist es invisible por si mismo, solo cobra vida atado a un input. El tercero es usar datalist para listas cerradas donde un select seria más claro, como "acepto terminos: si o no".

Un detalle de compatibilidad: datalist funciona en todos los navegadores modernos, pero si alguien usa un navegador muy antiguo, simplemente vera un campo de texto normal sin sugerencias. Eso se llama degradación elegante: la funcionalidad extra desaparece pero el formulario sigue usable. Es una de las bellezas de datalist frente a soluciones con JavaScript, que en navegadores viejos pueden romper todo el formulario.

optgroup: orden dentro de tus listas desplegables

Si el fieldset es el archivero de los formularios, el <optgroup> es el archivero de las listas desplegables. Cuando un <select> tiene muchas opciones, mostrarlas todas revueltas es como la pila de papeles del ejemplo anterior. El optgroup agrupa opciones relacionadas bajo un encabezado no seleccionable, creando secciones visuales dentro del desplegable. El usuario ve "Frutas" y "Verduras" como títulos, y debajo las opciones de cada categoría.

La sintaxis: dentro del select, abres <optgroup label="Nombre del grupo">, metes los <option> del grupo, y cierras </optgroup>. El atributo label es obligatorio: es el título visible del grupo. Mira un ejemplo de formulario de comida:


```

<label for="comida">Elige tu comida favorita:</label>
<select id="comida" name="comida">
  <optgroup label="Frutas">
    <option value="manzana">Manzana</option>
    <option value="platano">Platano</option>
    <option value="naranja">Naranja</option>
  </optgroup>
  <optgroup label="Verduras">
    <option value="zanahoria">Zanahoria</option>
    <option value="brocoli">Brocoli</option>
    <option value="espinaca">Espinaca</option>
  </optgroup>
  <optgroup label="Postres">
    <option value="pastel">Pastel</option>
    <option value="helado">Helado</option>
  </optgroup>
</select>

```

El navegador muestra los títulos de grupo en un estilo distinto (normalmente en negritas y sin poder seleccionarlos) y las opciones indentadas debajo. El usuario entiende la estructura de un vistazo. Un ejemplo más profesional: un formulario de registro a un curso donde las opciones se agrupan por nivel:

```

<select id="curso" name="curso">
  <optgroup label="Nivel principiante">
    <option value="html">HTML5 desde cero</option>
    <option value="css">CSS3 desde cero</option>
  </optgroup>
  <optgroup label="Nivel intermedio">
    <option value="js">JavaScript intermedio</option>
    <option value="python">Python intermedio</option>
  </optgroup>
  <optgroup label="Nivel avanzado">
    <option value="react">React avanzado</option>
    <option value="node">Node.js avanzado</option>
  </optgroup>
</select>

```

El `optgroup` también acepta el atributo `disabled`, que desactiva todo el grupo de un golpe. Útil para casos como "cursos agotados": agrupas los cursos sin cupo en un `optgroup disabled` y el usuario los ve pero no puede elegirlos. Es una forma honesta y clara de comunicar disponibilidad.

Errores comunes: olvidar el atributo label deja un grupo sin título, que se ve roto. Anidar optgroup dentro de optgroup no está permitido: el estándar solo admite un nivel, y los navegadores ignoran el anidado. Y el clásico: usar optgroup con solo dos o tres opciones totales; si la lista es corta, los grupos sobran y solo agregan ruido visual. Como regla, considera optgroup cuando el select pasa de ocho o diez opciones y esas opciones tienen categorías naturales.

Select múltiple: cuando una sola respuesta no basta

El <select> normal es de elección única: eliges una opción y listo. Pero hay preguntas donde la respuesta honesta es "varias": que lenguajes sabes, que días puedes asistir, que temas te interesan. Para eso existe el atributo multiple, que convierte el desplegable en una lista donde se pueden elegir varias opciones a la vez. Y casi siempre lo acompaña el atributo size, que dice cuantas opciones se muestran visibles sin tener que abrir nada.

```
<label for="lenguajes">Que lenguajes conoces? (manten Ctrl presionado para elegir varios)</label>
<select id="lenguajes" name="lenguajes" multiple size="5">
  <option value="html">HTML</option>
  <option value="css">CSS</option>
  <option value="js">JavaScript</option>
  <option value="python">Python</option>
  <option value="java">Java</option>
  <option value="php">PHP</option>
</select>
```

Con multiple y size="5", el control se muestra como una cajita con cinco opciones visibles y barra de desplazamiento para el resto. El usuario selecciona varias manteniendo presionada la tecla Ctrl (o Cmd en Mac) mientras hace clic. Aquí hay una lección importante de experiencia de usuario: ese mecanismo de Ctrl más clic no es obvio para todos. Mucha gente no sabe que existe. Por eso, cuando uses un select multiple, agrega una pequeña instrucción junto a la etiqueta, como en el ejemplo. Un control potente que nadie sabe usar es un control roto.

Alternativa honesta: para selecciones múltiples, muchos formularios modernos prefieren un grupo de casillas <input type="checkbox">, una por opción, porque son más obvias: se marcan con un clic, sin teclas especiales. El select multiple brilla cuando la lista es larga y el espacio es corto: veinte opciones en una cajita compacta contra veinte casillas ocupando media página. Conoce ambas herramientas y elige según el caso: pocas opciones, checkboxes; muchas opciones y poco espacio, select multiple.

Detalle tecnico: cuando el formulario se envía, las opciones elegidas en un select multiple viajan como varios valores con el mismo nombre. Del lado del servidor se reciben como una lista. No necesitas hacer nada especial en el HTML para que eso pase; el navegador lo maneja solo. Solo asegurate de que el atributo name este presente, porque sin name los datos ni siquiera se envían.

textarea: control total con filas y columnas

Ya conociste el `<input type="text">` para textos cortos: nombres, correos, ciudades. Pero cuando el usuario necesita escribir un párrafo, una descripción, un mensaje o un comentario, el campo de una sola línea se queda cortísimo. Para eso existe `<textarea>`: un area de texto de varias líneas, redimensionable, pensada para contenido largo. La diferencia no es solo visual: semánticamente le dices al navegador "aquí viene un texto extenso", y el navegador responde con un control adecuado.

A diferencia de input, que es una etiqueta autocontenida, textarea tiene etiqueta de apertura y cierre, y el contenido inicial va entre ellas:

```
<label for="mensaje">Cuentanos tu idea:</label>
<textarea id="mensaje" name="mensaje" rows="6" cols="50" placeholder="Describe tu
proyecto en pocas lineas..."></textarea>
```

Los atributos rows y cols controlan el tamaño visible: rows es el número de líneas visibles de alto, cols es el ancho aproximado en caracteres. Ojo con la palabra "aproximado": cols no es una medida exacta en pixeles, es un promedio basado en el ancho de los caracteres de la fuente. En la práctica, rows="6" cols="50" te da un area cómoda para un mensaje mediano. Y recuerda: rows y cols definen el tamaño visible, no un límite de lo que el usuario puede escribir; el area crece con barra de desplazamiento si el texto se pasa.

Otros atributos útiles: maxlength limita el número de caracteres (perfecto para "máximo 500 caracteres" en una biografia), placeholder muestra un texto de ayuda que desaparece al escribir, wrap controla como se envían los saltos de línea (con wrap="hard" se envían tal cual se ven), readonly lo vuelve de solo lectura y required lo hace obligatorio. Combinados, te dan control fino sin una sola línea de JavaScript:

```
<label for="bio">Tu biografia (maximo 200 caracteres):</label>
<textarea id="bio" name="bio" rows="4" cols="50" maxlength="200"
placeholder="Cuentanos quien eres y que haces..." required></textarea>
```

Errores comunes con textarea. El primero y más tonto: escribir `<textarea />` como si fuera autocontenida, al estilo de `<input>`. No lo es: necesita su etiqueta de cierre, y si la omites el navegador se traga todo el contenido siguiente como si fuera texto del area. El segundo: poner saltos de línea o espacios entre `<textarea>` y `</textarea>` "para que se vea ordenado" en el código. Esos espacios se convierten en contenido inicial del campo, y luego el placeholder no aparece porque técnicamente el campo ya tiene contenido (espacios invisibles). Si no quieres texto inicial, pega las etiquetas sin nada entre ellas. El tercero: usar textarea para todo, incluso para el nombre o el correo. Un campo gigante para escribir "Juan" confunde al usuario; cada control tiene su proposito y usar el adecuado es parte del profesionalismo.

Buena práctica general de formularios profesionales: combina todo lo aprendido. Un formulario de registro pro típico usa fieldset para agrupar (datos personales, dirección, preferencias), datalist para la ciudad, optgroup en el select de curso, y textarea para comentarios adicionales. Cuando veas el primer ejemplo de la sección de ejemplos, vas a reconocer cada pieza en su lugar, trabajando en equipo.

Etiquetas obsoletas: que significa que una etiqueta muera

En HTML, "obsoleto" no significa "funciona mal": significa "el estandar ya no la respalda y puede dejar de funcionar en cualquier momento". Es como un modelo de auto discontinuado: todavía arranca y camina, pero ya no hay refacciones, nadie lo recomienda y cada año que pasa es más riesgoso usarlo. Las etiquetas obsoletas (en ingles, deprecated u obsolete) fueron útiles en su epoca, pero el lenguaje evoluciona y hoy existen formas mejores, más limpias y más potentes de lograr lo mismo.

Por que mueren las etiquetas. La razon de fondo es una idea que vas a escuchar toda tu carrera: separación de responsabilidades. HTML se encarga de la estructura y el significado del contenido (que es un título, que es un párrafo, que es un formulario). CSS se encarga de la presentación (colores, tamanos, posiciones, animaciones). JavaScript se encarga del comportamiento (que pasa cuando haces clic). Cuando HTML intentaba hacer el trabajo de CSS, el resultado era código mezclado, difícil de mantener y feo por dentro. Las etiquetas de presentación como `<font>` o `<center>` mezclaban el "que se ve" dentro del "que es", y por eso el estandar las jubilo. Hoy la regla es clara: si es apariencia, va en CSS; si es estructura, va en HTML.

Como reconocer una etiqueta obsoleta. Primera señal: su único proposito es visual (centrar, poner color, hacer marquesina) sin aportar significado. Segunda señal: los tutoriales que la enseñan tienen diseno de hace quince años o hablan de "Internet Explorer". Tercera señal: los validadores de HTML la marcan con advertencias. Y la cuarta, la más práctica: si la etiqueta no

aparece en la documentación actual de MDN como elemento válido, no la uses. Desarrollar el instinto para oler código viejo es una habilidad profesional tan valiosa como aprender etiquetas nuevas.

<center>: la etiqueta que quería centrarlo todo

La etiqueta <center> hacia exactamente lo que su nombre promete: centraba todo lo que contenía. Texto, imágenes, tablas, lo que fuera quedaba centrado en la página. En los noventa era la forma estándar de centrar cosas, y todavía hoy la encuentras en código viejo y en tutoriales desactualizados. El problema: centrar es presentación pura, no estructura. Decir "esto es un título" es estructura; decir "esto va centrado" es apariencia. Por eso <center> murio: su trabajo le pertenece a CSS.

La forma moderna de centrar texto es la propiedad text-align de CSS:

```
<!-- Código viejo, NO lo uses -->
<center><h2>Bienvenidos a mi pagina</h2></center>

<!-- Forma moderna: el HTML dice QUE es, el CSS dice COMO se ve -->
<h2 style="text-align: center;">Bienvenidos a mi pagina</h2>
```

Y para centrar un bloque completo (como una caja o una imagen) de forma moderna se usan márgenes automáticos:

```
<!-- Código viejo, NO lo uses -->
<center></center>

<!-- Forma moderna -->

```

No te preocupes si la sintaxis CSS todavía te parece extraña: CSS es el siguiente paso natural después de HTML y lo dominarás a su tiempo. Lo importante ahora es el hábito mental: cuando sientas la tentación de buscar "una etiqueta para centrar", recuerda que esa etiqueta no debe existir en tu código. El HTML describe, el CSS decora. Cada vez que respetas esa división, tu código se vuelve más limpio, más fácil de mantener y más profesional.

: cuando el texto gritaba con etiquetas

La etiqueta permitía cambiar la apariencia del texto directamente en el HTML, con atributos como size (tamaño del 1 al 7), color (nombre o código hexadecimal) y face (tipo de letra). Un texto rojo, grande y en Arial se escribía así:

```
<!-- Código viejo, NO lo uses -->
<font size="5" color="red" face="Arial">Oferta especial, solo hoy</font>

<!-- Forma moderna -->
<p style="font-size: 24px; color: red; font-family: Arial;">Oferta especial, solo hoy</p>
```

Parece práctico, verdad? El problema aparece cuando tu página crece. Imagina cien páginas con esparcidos por todas partes, y un día el cliente dice "el rojo ya no, ahora todo en azul". Con tendrías que cazar y cambiar cien etiquetas una por una. Con CSS cambias una sola línea y las cien páginas se actualizan al instante. Esa es la diferencia entre código mantenible y código que se vuelve una pesadilla: la presentación centralizada contra la presentación regada por todo el documento.

Además, tenía limitaciones ridículas: solo siete tamaños posibles, sin control fino, sin responsive, sin nada. CSS te da control total: tamaños en píxeles, porcentajes, unidades relativas, colores con transparencia, cientos de fuentes, y reglas que se aplican a toda la página desde un solo lugar. Usar hoy es como querer enviar un mensaje urgente por carta cuando tienes teléfono: técnicamente llega, pero no tiene sentido.

<marquee>: el texto que no se quedaba quieto

La etiqueta <marquee> hacía que el texto se moviera por la pantalla como un letrero luminoso: de derecha a izquierda, de izquierda a derecha, rebotando. En los noventa era la sensación: toda página personal que se respetara tenía su texto en movimiento. Hoy es el ejemplo perfecto de una etiqueta que murió por buenas razones, y vale la pena entenderlas porque enseñan mucho sobre diseño web moderno.

Primera razón: accesibilidad. El texto en movimiento es un problema real para personas con trastornos de atención, dislexia o sensibilidad al movimiento: les dificulta leer y puede causar mareos o náuseas. Una web profesional no debe agredir a sus visitantes. Segunda razón: control. El <marquee> se movía a su ritmo, sin que el usuario pudiera pausarlo, y eso viola un principio básico de usabilidad: el usuario manda, no la página. Tercera razón: era presentación y comportamiento metidos en HTML, doble pecado contra la separación de responsabilidades.

La forma moderna de animar texto es CSS, que permite animaciones suaves, pausables y controlables:

```
<!-- Codigo viejo, NO lo uses -->
<marquee>Bienvenidos a mi pagina personal</marquee>

<!-- Forma moderna: HTML limpio, la animacion vive en CSS -->
<p class="anuncio">Bienvenidos a mi pagina personal</p>
```

(La clase "anuncio" se animaría desde una hoja de estilos CSS con reglas de animación, incluyendo la opción de pausar el movimiento, algo que <marquee> jamás permitio hacer bien.) La lección de <marquee> va más allá de la etiqueta: en diseño web, el movimiento debe tener un propósito y respetar al usuario. Un texto que se mueve solo porque se puede es ruido; una animación sutil que guía la atención es diseño.

 e <i> contra y : no murieron, pero se usan mal

Aquí hay un matiz importante: (bold, negritas) e <i> (italic, cursiva) no están obsoletas. Siguen siendo válidas en HTML5. El problema es que la mayoría de los principiantes las usa para lo que no son, y entender la diferencia te pone por encima del montón.

La clave está en el significado, no en la apariencia. significa "esto es importante": el navegador lo muestra en negritas, pero lo crucial es que los lectores de pantalla lo anuncian con énfasis y los buscadores le dan más peso. significa "esto lleva énfasis": se muestra en cursiva, pero semánticamente indica entonación, como cuando en una conversación enfatizas una palabra. En cambio, solo dice "muéstrame esto en negritas" sin ningún significado especial (útil para nombres de productos o palabras clave donde solo importa lo visual), e <i> solo dice "muéstrame esto en cursiva" (útil para términos técnicos, palabras en otro idioma o pensamientos).

Comparalo así:

```
<!-- Solo apariencia, sin significado especial -->
<p>El producto <b>SuperWidget 3000</b> ya está disponible.</p>
<p>La palabra <i>software</i> viene del inglés.</p>

<!-- Con significado: importancia y énfasis reales -->
<p><strong>Atención:</strong> la oferta termina <em>hoy mismo</em>.</p>
```

En el primer caso, el nombre del producto en negritas es una decisión visual y la palabra en ingles en cursiva es una convención tipografica: `<b>` e `<i>` estan bien usadas. En el segundo caso, "Atención" es realmente importante y "hoy mismo" lleva énfasis real en la lectura: `<strong>` y `<em>` son las correctas. Si un lector de pantalla lee tu página, la diferencia se escucha: con `strong` y `em` la voz enfatiza; con `b` e `i`, no pasa nada especial.

Error común: usar `<b>` para todo lo importante porque "se ve igual". Se ve igual en tu pantalla, pero por dentro tu página es semánticamente pobre: ni los buscadores ni las tecnologías de asistencia saben que ese texto era importante. Y al revés: usar `<strong>` solo porque quieres negritas visuales en algo sin importancia real. Preguntate siempre: si alguien escuchara esta página en lugar de verla, esta palabra debería sonar enfatizada? Si la respuesta es si, usa `strong` o `em`. Si solo quieres que se vea distinto, `b` o `i` estan bien, o mejor aun, CSS.

Por que murieron y que usar en su lugar: HTML es estructura, no estilo

Juntando las piezas, la historia de las etiquetas obsoletas es la historia de HTML aprendiendo cual es su lugar. En los inicios, HTML hacia de todo porque no había otra cosa: CSS no existía o era muy limitado. Las páginas mezclaban contenido y diseño en un solo archivo, y funcionaba mientras las páginas eran simples. Pero la web creció, los sitios pasaron de cinco páginas a cinco mil, los equipos pasaron de una persona a cincuenta, y el modelo de "todo revuelto" colapso bajo su propio peso. La solución fue repartir el trabajo: HTML para la estructura, CSS para la presentación, JavaScript para el comportamiento. Las etiquetas que se negaron a soltar el trabajo de CSS fueron jubiladas.

Piensa en las etiquetas obsoletas como ropa pasada de moda. En los noventa, los pantalones acampanados eran lo máximo; hoy, si llegas a una entrevista de trabajo con unos, nadie te toma en serio aunque "todavía cubren las piernas". Con `<center>`, `<font>` y `<marquee>` pasa igual: todavía "funcionan" en el sentido de que el navegador las dibuja, pero delatan que quien escribió el código se quedó en otra época. Y en el mundo laboral del desarrollo web, verte actualizado es parte del trabajo: tu código es tu carta de presentación.

Que usar en su lugar, en resumen práctico:

- En lugar de `<center>`: CSS con `text-align: center` para texto, o `margin: auto` para bloques.
- En lugar de `<font>`: CSS con `font-family`, `font-size` y `color`.
- En lugar de `<marquee>`: animaciones CSS, que son pausables y respetan al usuario.
- En lugar de `<b>` para lo importante: `<strong>`. En lugar de `<i>` para énfasis real: `<em>`.
- En lugar de atributos de presentación en general (como `bgcolor` o `align`): CSS.

Y la regla de oro que resume toda la unidad: si lo que quieres lograr es que algo se vea de cierta forma, la respuesta está en CSS, no en buscar una etiqueta mágica de HTML. HTML responde a la pregunta "que es esto"; CSS responde a "como se ve". Cuando interiorices esa división, vas a escribir HTML más limpio que el 90 por ciento de los tutoriales viejos que circulan por internet, y vas a estar listo para aprender CSS con la mentalidad correcta.

Proyecto: planificando tu portafolio antes de escribir una línea

Todo proyecto serio empieza antes del código: empieza en papel, o en una nota, o en una servilleta. Un portafolio web personal no es la excepción. Antes de abrir tu editor, necesitas responder tres preguntas: quien eres profesionalmente, que quieres mostrar y a quien va dirigido. Un portafolio para buscar trabajo como desarrollador frontend no es igual a uno para ofrecer servicios freelance ni a uno de estudiante que muestra sus practicas. Define tu objetivo en una frase, por ejemplo: "mi portafolio debe convencer a un reclutador de que se programar páginas web modernas". Esa frase va a guiar cada decisión de aquí en adelante.

Que lleva un portafolio profesional. No hay una ley escrita, pero los portafolios que funcionan comparten los mismos bloques, y cada bloque responde a una pregunta que el visitante se hace:

- Encabezado con navegación: quien eres y a donde puedo ir dentro de tu página.
- Sección principal o de bienvenida: que haces, en una frase potente, con tu nombre bien visible.
- Sobre mi: quien eres en dos o tres párrafos honestos, que tecnologías manejas.
- Proyectos: la prueba de tu trabajo, con título, descripción breve y enlace a cada uno.
- Habilidades: tus tecnologías organizadas, claras y sin exagerar.
- Contacto: como te localizo, con un formulario o tus datos directos.
- Pie de página: derechos reservados y enlaces a tus redes profesionales.

Fijate que cada bloque mapea de forma natural a etiquetas semanticas que ya conoces: header, nav, main, section, article, footer. El HTML semantico de la unidad 8 y el proyecto de esta unidad se dan la mano: no aprendiste esas etiquetas por teoria, las aprendiste para usarlas aquí.

Planifica también el contenido real, no solo la estructura. Escribe en un documento aparte los textos de cada sección: tu presentación de dos líneas, tu párrafo de "sobre mi", la lista de tus proyectos (aunque al inicio sean practicas del curso: valen, porque demuestran progreso), tus

habilidades honestas. Escribir los textos antes del código tiene una ventaja enorme: cuando llegues al HTML solo te preocupas por la estructura, y no te bloqueas mirando una página vacía sin saber que escribir. Los profesionales separan contenido y código igual que separan HTML y CSS: primero que decir, luego como estructurarlo.

Header con logo y navegación: la puerta de entrada

El <header> es lo primero que ve el visitante, y tiene un trabajo doble: identidad y navegación. Identidad significa que en un segundo quede claro de quien es la página: tu nombre o tu marca como logo, visible y memorable. Navegación significa que el visitante pueda moverse por tu página sin perderse: un <nav> con enlaces a cada sección importante. Piensa en el header como la fachada de una tienda: el letrero con el nombre y la puerta bien señalizada. Si la fachada es confusa, la gente no entra.

Estructura típica de un header de portafolio:

```
<header>
  <div class="logo">
    
    <span>Esteban Ega</span>
  </div>
  <nav>
    <ul>
      <li><a href="#inicio">Inicio</a></li>
      <li><a href="#sobre-mi">Sobre mi</a></li>
      <li><a href="#proyectos">Proyectos</a></li>
      <li><a href="#contacto">Contacto</a></li>
    </ul>
  </nav>
</header>
```

Observa las decisiones: el logo combina imagen y nombre dentro de un div, porque el logo es una unidad visual. La navegación usa una lista desordenada con enlaces, que es la forma semantica correcta de un menú: no es solo una fila de enlaces sueltos, es una lista de opciones de navegación, y los lectores de pantalla la anuncian como tal. Los enlaces apuntan a anclas (#inicio, #sobre-mi): cuando las secciones tengan esos id, hacer clic llevara al visitante a cada parte de la página sin recargarla. Es la navegación clásica de los portafolios de una sola página.

Buenas practicas del header: mantenlo simple, con cuatro a seis enlaces máximo; más enlaces confunden. El logo debe llevar siempre un alt descriptivo. Usa los mismos textos en el

menú que en los títulos de las secciones, para que el visitante sienta coherencia. Y un detalle profesional: el orden del menú debe seguir el orden de las secciones en la página; si el menú dice Inicio, Sobre mi, Proyectos, Contacto, las secciones deben aparecer en ese mismo orden.

Main con el contenido principal: el corazón de tu página

Si el header es la fachada, el <main> es el interior de la tienda: donde está todo lo que el visitante vino a ver. La regla es estricta y sencilla: debe haber un solo <main> por página, y dentro va el contenido principal, el que justifica la existencia de la página. Todo lo que no sea contenido principal (navegación repetida, publicidad, barras laterales secundarias) va fuera. Esta regla no es capricho: los lectores de pantalla tienen un atajo para saltar directo al main, y si metes cosas secundarias adentro, les arruinas la navegación.

Dentro del main, organiza con <section> por bloques temáticos y <article> para piezas independientes. En tu portafolio se verá así:

```

<main>
  <section id="inicio">
    <h1>Hola, soy Esteban Ega</h1>
    <p>Desarrollador web: convierto ideas en paginas rapidas y claras.</p>
    <a href="#proyectos">Ver mis proyectos</a>
  </section>
  <section id="sobre-mi">
    <h2>Sobre mi</h2>
    <p>...</p>
  </section>
  <section id="proyectos">
    <h2>Mis proyectos</h2>
    <article>
      <h3>Curso completo de HTML5</h3>
      <p>...</p>
      <a href="#">Ver proyecto</a>
    </article>
    <article>
      <h3>...</h3>
      <p>...</p>
    </article>
  </section>
  <section id="contacto">
    <h2>Contacto</h2>
    <form>...</form>
  </section>
</main>

```

Cada section tiene su id, que coincide con los enlaces del menú: así la navegación por anclas funciona. Cada section abre con su encabezado (h1 una sola vez para el título principal, h2 para las secciones, h3 para los proyectos individuales): la jerarquía de encabezados es el índice de tu página. Los proyectos van en <article> porque cada proyecto es una pieza autocontenida: podría publicarse por separado y seguiría teniendo sentido. Y la sección de contacto albergara el formulario profesional que aprendiste a construir en esta misma unidad: ves como todo se conecta.

Error común: convertir el main en un basurero donde va "todo lo demás". Si tienes dudas sobre si algo va en el main, pregúntate: si le quito esto a la página, la página pierde su razón de ser? Si la respuesta es no, probablemente va en otro lado (un aside, el footer, o no va). Otro error: olvidar los id en las secciones y luego preguntarse por qué los enlaces del menú no llevan a ningún lado. Los id son el pegamento entre el nav y las secciones: sin ellos, el menú es decoración.

Footer con derechos: el cierre profesional

El <footer> es el final de la página y, aunque muchos lo tratan como un trámite, es una pieza de profesionalismo. Un footer bien hecho transmite seriedad; uno descuidado o ausente deja la sensación de página a medias. Piensa en el footer como la contraportada de un libro: breve, pero con la información legal y de contacto que el lector espera encontrar.

Que lleva un footer de portafolio:

- Aviso de derechos reservados con el año y tu nombre: el clásico "Todos los derechos reservados".
- Enlaces a tus redes profesionales o tu correo.
- Opcional: un mini menú de navegación repetido, útil en páginas largas.
- Opcional: una frase breve que refuerce tu marca personal.

```
<footer>
  <p>&copy; 2026 Esteban Ega. Todos los derechos reservados.</p>
  <nav>
    <ul>
      <li><a href="#inicio">Inicio</a></li>
      <li><a href="#proyectos">Proyectos</a></li>
      <li><a href="#contacto">Contacto</a></li>
    </ul>
  </nav>
  <p>Hecho con HTML, CSS y muchas ganas de aprender.</p>
</footer>
```

El símbolo © es una entidad HTML que muestra el símbolo de copyright sin problemas de codificación: usalo con confianza. El año debe ser el actual; un footer que dice 2019 en 2026 grita abandono, así que acostúmbrate a actualizarlo. Y nota que el footer puede contener su propio <nav>: es perfectamente válido tener más de un elemento nav en la página, siempre que cada uno tenga un propósito claro (el del header es la navegación principal, el del footer es secundaria).

Organización de archivos y carpetas: tu proyecto como un profesional

Un proyecto web no es un archivo suelto en el escritorio: es una carpeta organizada con una lógica que cualquier desarrollador entiende al abrirla. Adoptar una estructura profesional desde tu primer proyecto te ahorra caos futuro y, cuando trabajes en equipo, tus compañeros te lo van a agradecer. Esta es la estructura base recomendada para tu portafolio:

```
mi-portafolio/  
  index.html  
  css/  
    estilos.css  
  img/  
    logo.png  
    foto-perfil.jpg  
    proyecto-1.jpg  
  js/  
    scripts.js  
  docs/  
    curriculum.pdf
```

Cada carpeta tiene su razón: index.html es la puerta de entrada (los servidores la buscan por defecto cuando alguien visita tu dominio). La carpeta css guardará tus hojas de estilo cuando aprendas CSS. La carpeta img centraliza todas las imágenes, para no tener archivos regados. La carpeta js guardará tus scripts cuando aprendas JavaScript. La carpeta docs es para descargables como tu curriculum en PDF. Esta convención de nombres en minúsculas, sin espacios ni caracteres raros, evita el 90 por ciento de los errores de "no se muestra mi imagen": la mayoría de esos errores son rutas mal escritas o nombres con mayúsculas que en el servidor no coinciden.

Reglas de oro para las rutas: usa rutas relativas (img/logo.png) en lugar de rutas absolutas de tu computadora (C:/usuarios/...), porque las absolutas se rompen en cuanto subes el proyecto a internet. Mantén los nombres de archivo cortos, en minúsculas, sin espacios (usa guiones: foto-perfil.jpg, no "Foto de perfil FINAL 2.jpg"). Y crea la estructura de carpetas antes de

escribir código: primero el archivero, luego los papeles. Es el mismo principio del fieldset aplicado a tu disco duro.

La estructura base como cimientos: la analogía que lo amarra todo

Volvamos a la analogía de la construcción, porque resume la unidad entera. Tu portafolio es un edificio. La planificación (que secciones lleva, que textos, a quien va dirigido) es el plano del arquitecto: sin plano, los albaniles improvisan y el edificio sale chueco. La estructura HTML base (header, nav, main, section, footer) son los cimientos y el esqueleto de concreto: no se ven tan espectaculares como la fachada, pero sostienen todo. La organización de carpetas es la bodega de materiales ordenada: cuando necesitas un ladrillo, sabes en que pasillo esta. Los formularios profesionales son las instalaciones electricas y de plomeria: el visitante no las ve, pero si fallan, la casa no sirve. Y las etiquetas obsoletas son materiales de construcción vencidos: todavía parecen servir, pero un buen arquitecto jamás los usaría en una obra nueva.

En las proximas unidades vas a levantar los pisos: más secciones, el formulario de contacto completo, detalles semanticos, y eventualmente el diseno con CSS. Pero nada de eso funcionara bien si los cimientos estan mal. Por eso esta unidad insiste tanto en la base: un header bien estructurado, un main con un solo proposito, un footer completo y carpetas ordenadas. Cuando termines el curso y mires tu portafolio terminado, vas a ver una construcción sólida, y todo empezo aquí, con estas decisiones aparentemente simples.

Errores comunes de la unidad (y como evitarlos)

- Agrupar con fieldset cada campo por separado, creando un recuadro por campo. Agrupa por temas, no por campos.
- Olvidar el legend dentro del fieldset, dejando grupos sin título.
- Que el atributo list del input no coincida exactamente con el id del datalist, y las sugerencias nunca aparezcan sin ningún error visible.
- Poner espacios o saltos de línea entre <textarea> y </textarea>, que se convierten en contenido invisible y anulan el placeholder.
- Escribir <textarea /> sin etiqueta de cierre, haciendo que el navegador se trague el contenido siguiente.
- Usar <center>, o <marquee> porque "asi lo vi en un tutorial": verifica la fecha del tutorial y la documentación actual antes de copiar.

- Usar `<b>` para todo lo importante en lugar de `<strong>`, empobreciendo semánticamente la página.
- Poner más de un `<main>` en la página, o meter contenido secundario dentro del único `main`.
- Olvidar los `id` en las secciones, rompiendo la navegación por anclas del menú.
- Guardar el proyecto como archivos sueltos con nombres como "página final 2.html": usa carpetas y nombres en minúsculas con guiones.
- Usar rutas absolutas de tu computadora en `src` y `href`, que se rompen al publicar la página.

Buenas practicas que te distinguen como profesional

- En formularios largos, agrupa siempre con `fieldset` y `legend`: el usuario percibe orden y tu código queda documentado solo.
- Elige `datalist` para listas largas o abiertas, `select` con `optgroup` para listas cerradas con categorías, y `checkboxes` para selecciones múltiples cortas y obvias.
- Cuando uses `select multiple`, agrega una instrucción breve sobre como elegir varias opciones: no asumas que el usuario lo sabe.
- Dimensiona los `textarea` con `rows` y `cols` según el contenido esperado: ni una línea para un mensaje largo, ni un campo gigante para un nombre.
- Trata el HTML como estructura y nada más: cada vez que quieras cambiar apariencia, piensa en CSS, no en etiquetas nuevas.
- Audita tu código buscando etiquetas obsoletas antes de darlo por terminado, igual que revisas la ortografía de un texto importante.
- Usa `<strong>` y `<em>` cuando el significado lo pida, no solo cuando quieras negritas o cursiva visual.
- Planifica el portafolio en papel antes del código: secciones, textos y objetivo en una frase.
- Un solo `<main>` por página, un `<h1>` por página, y una jerarquía de encabezados que funcione como índice.
- Crea la estructura de carpetas primero y respeta los nombres en minúsculas con guiones: tu yo del futuro te lo agradecerá.
- Actualiza el año del footer y revisa que todos los enlaces del menú apunten a secciones que existen.

Ejemplos paso a paso

Antes de entrar a los ejemplos, un recordatorio de como estudiarlos: no te limites a leer el código en la pantalla. Copia cada ejemplo a tu editor, guardalo como archivo html y abrelo en el navegador. Luego juega con el: cambia los textos, agrega un campo más al formulario, quita un legend para ver como se ve el grupo sin título, escribe mal un id a proposito para comprobar que el datalist deja de funcionar. Romper cosas a proposito es una de las formas más rápidas de aprender, porque el error te ensena lo que la lectura no puede: la consecuencia real de cada decisión. Los cuatro ejemplos estan ordenados de menor a mayor integración: primero el formulario profesional aislado, luego la comparativa de código viejo contra moderno, después la estructura completa del portafolio donde todo convive, y al final el footer reutilizable que llevaras a todos tus proyectos.

Ejemplo 1: formulario de registro profesional con fieldset, datalist y optgroup

Este es el ejemplo estrella de la unidad: un formulario de registro a un curso en línea que combina todo lo aprendido. Copialo, abrelo en tu navegador y observa como cada pieza cumple su función. Los comentarios explican cada decisión.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Registro al curso - Lucas EGA Academy</title>
</head>
<body>

  <h1>Registrate gratis</h1>
  <p>Completa el formulario y empieza a aprender hoy mismo.</p>

  <form action="/registrar" method="post">

    <!-- GRUPO 1: datos personales, todo lo que identifica al usuario -->
    <fieldset>
      <legend>Datos personales</legend>

      <label for="nombre">Nombre completo:</label>
      <input type="text" id="nombre" name="nombre" required>
      <br><br>

      <label for="email">Correo electronico:</label>
      <input type="email" id="email" name="email" required>
```

```

<br><br>

<!-- datalist: sugerencias de ciudad, pero el usuario puede escribir otra -->
<label for="ciudad">Ciudad:</label>
    <input type="text" id="ciudad" name="ciudad" list="lista-ciudades"
placeholder="Empieza a escribir...">
    <datalist id="lista-ciudades">
        <option value="Ciudad de Mexico">
        <option value="Guadalajara">
        <option value="Monterrey">
        <option value="Puebla">
        <option value="Merida">
        <option value="Queretaro">
    </datalist>
</fieldset>
<br>

<!-- GRUPO 2: que quiere estudiar, con opciones organizadas por nivel -->
<fieldset>
    <legend>Tu curso</legend>

    <label for="curso">Que curso te interesa?</label>
    <select id="curso" name="curso" required>
        <option value="">-- Elige un curso --</option>
        <optgroup label="Nivel principiante">
            <option value="html">HTML5 desde cero</option>
            <option value="css">CSS3 desde cero</option>
        </optgroup>
        <optgroup label="Nivel intermedio">
            <option value="js">JavaScript intermedio</option>
            <option value="python">Python intermedio</option>
        </optgroup>
        <optgroup label="Nivel avanzado">
            <option value="react">React avanzado</option>
            <option value="node">Node.js avanzado</option>
        </optgroup>
    </select>
<br><br>

<!-- select multiple: varios intereses, con instruccion clara -->
    <label for="intereses">Temas que te interesan (manten Ctrl para elegir
varios):</label><br>
    <select id="intereses" name="intereses" multiple size="4">
        <option value="web">Desarrollo web</option>

```

```

        <option value="movil">Apps moviles</option>
        <option value="datos">Ciencia de datos</option>
        <option value="juegos">Videojuegos</option>
        <option value="ia">Inteligencia artificial</option>
    </select>
</fieldset>
<br>

<!-- GRUPO 3: espacio libre para que el usuario cuente su objetivo -->
<fieldset>
    <legend>Comentarios</legend>

    <label for="objetivo">Cual es tu objetivo aprendiendo programacion?</label><br>
    <textarea id="objetivo" name="objetivo" rows="5" cols="50" maxlength="300"
placeholder="Por ejemplo: quiero conseguir mi primer trabajo como
desarrollador..."></textarea>
</fieldset>
<br>

<button type="submit">Crear mi cuenta gratis</button>

</form>

</body>
</html>

```

Que observar en este ejemplo. Primero, los tres fieldset dividen el formulario en bloques mentales claros: quien eres, que quieres estudiar y que quieres contar. El usuario nunca se siente abrumado porque solo procesa un grupo a la vez. Segundo, el datalist de ciudades ayuda sin obligar: si tu ciudad no esta, la escribes y listo. Tercero, el optgroup convierte una lista de seis cursos en tres categorías fáciles de escanear. Cuarto, el select multiple incluye la instrucción de la tecla Ctrl, un detalle de profesional que evita frustración. Quinto, el textarea tiene rows, cols, maxlength y placeholder trabajando juntos: tamaño comodo, limite claro y ejemplo de que escribir. Todo el formulario se puede usar solo con el teclado, funciona sin JavaScript y se entiende sin explicaciones extra. Eso es un formulario profesional.

Ejemplo 2: comparativa etiqueta obsoleta contra forma moderna

Nada ensena mejor que ver el antes y el después lado a lado. Este ejemplo muestra los tres pecados clasicos y su redención moderna. Estudia cada pareja: a la izquierda lo que NO debes hacer, a la derecha lo que SI.

```

<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>Antes y despues: codigo moderno</title>
</head>
<body>

  <!-- CASO 1: centrar un titulo -->
  <!-- ANTES (obsoleto, NO usar): -->
  <!-- <center><h2>Ofertas de la semana</h2></center> -->
  <!-- DESPUES (moderno): el HTML dice QUE es, el CSS dice COMO se ve -->
  <h2 style="text-align: center;">Ofertas de la semana</h2>

  <!-- CASO 2: texto con color y tamano -->
  <!-- ANTES (obsoleto, NO usar): -->
  <!-- <font size="4" color="blue" face="Arial">Envio gratis en todos los
cursos</font> -->
  <!-- DESPUES (moderno): -->
  <p style="font-size: 18px; color: blue; font-family: Arial;">Envio gratis en
todos los cursos</p>

  <!-- CASO 3: texto en movimiento -->
  <!-- ANTES (obsoleto, NO usar): -->
  <!-- <marquee>Incripciones abiertas, aparta tu lugar</marquee> -->
  <!-- DESPUES (moderno): texto quieto y legible; si se quiere animacion, se hace
con CSS -->
  <p><strong>Incripciones abiertas:</strong> aparta tu lugar hoy mismo.</p>

  <!-- CASO 4: negritas con significado contra negritas solo visuales -->
  <!-- Solo visual (correcto con b): el nombre del producto no es "importante",
solo destaca -->
  <p>El curso <b>HTML5 desde cero</b> incluye 30 clases en video.</p>
  <!-- Con significado real (correcto con strong y em): aqui SI hay importancia y
enfasis -->
  <p><strong>Importante:</strong> la promocion termina <em>este viernes</em>.</p>

</body>
</html>

```

La lección de este ejemplo no es memorizar cuatro reemplazos: es interiorizar el criterio. Cada vez que veas una etiqueta vieja en un tutorial, pregúntate "esto describe estructura o solo cambia la apariencia?". Si solo cambia la apariencia, la respuesta moderna vive en CSS. Con

el tiempo, este criterio se vuelve automático y vas a detectar código obsoleto al primer vistazo, como un mecanico que escucha un motor y sabe que falla sin abrir el cofre.

Ejemplo 3: estructura base completa del portafolio

Aquí esta el esqueleto de tu proyecto final: una página de portafolio de una sola hoja con header, navegación por anclas, main con cuatro secciones y footer. Es la base sobre la que vas a construir en las siguientes unidades. Guardalo como index.html dentro de tu carpeta de proyecto.

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Esteban Ega - Desarrollador Web</title>
</head>
<body>

  <!-- CABECERA: identidad + navegacion principal -->
  <header>
    <div class="logo">
      
      <span>Esteban Ega</span>
    </div>
    <nav>
      <ul>
        <li><a href="#inicio">Inicio</a></li>
        <li><a href="#sobre-mi">Sobre mi</a></li>
        <li><a href="#proyectos">Proyectos</a></li>
        <li><a href="#contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>

  <!-- CONTENIDO PRINCIPAL: un solo main, secciones con id para las anclas -->
  <main>

    <section id="inicio">
      <h1>Hola, soy Esteban Ega</h1>
      <p>Desarrollador web en formacion: convierto ideas en paginas rapidas, claras y bien estructuradas.</p>
      <a href="#proyectos">Ver mis proyectos</a>
```

```

</section>

<section id="sobre-mi">
  <h2>Sobre mi</h2>
  
  <p>Soy estudiante de desarrollo web en Lucas EGA Academy. Me apasiona el código
limpio y las páginas que se entienden solas.</p>
  <h3>Mis habilidades</h3>
  <ul>
    <li>HTML5 semántico</li>
    <li>Formularios profesionales</li>
    <li>Estructura de proyectos web</li>
  </ul>
</section>

<section id="proyectos">
  <h2>Mis proyectos</h2>

  <article>
    <h3>Portafolio personal</h3>
    
    <p>Mi página personal construida con HTML5 semántico: header, navegación por
anclas y formulario de contacto.</p>
    <a href="#">Ver proyecto</a>
  </article>

  <article>
    <h3>Formulario de registro pro</h3>
    
    <p>Formulario con fieldset, datalist y optgroup: organización profesional de
campos.</p>
    <a href="#">Ver proyecto</a>
  </article>

</section>

<section id="contacto">
  <h2>Contacto</h2>
  <p>Tienes un proyecto en mente? Escribeme.</p>
  <form action="/contacto" method="post">
    <fieldset>
      <legend>Tus datos</legend>
      <label for="c-nombre">Nombre:</label>

```

```

        <input type="text" id="c-nombre" name="nombre" required>
        <br><br>
        <label for="c-email">Correo:</label>
        <input type="email" id="c-email" name="email" required>
    </fieldset>
    <br>
    <label for="c-mensaje">Mensaje:</label><br>
        <textarea id="c-mensaje" name="mensaje" rows="5" cols="50"
placeholder="Cuentame tu idea..." required></textarea>
        <br><br>
        <button type="submit">Enviar mensaje</button>
    </form>
</section>

</main>

<!-- PIE DE PAGINA: derechos, navegacion secundaria y firma -->
<footer>
    <p>&copy; 2026 Esteban Ega. Todos los derechos reservados.</p>
    <nav>
        <ul>
            <li><a href="#inicio">Inicio</a></li>
            <li><a href="#proyectos">Proyectos</a></li>
            <li><a href="#contacto">Contacto</a></li>
        </ul>
    </nav>
    <p>Hecho con HTML5 semantico y muchas ganas de aprender.</p>
</footer>

</body>
</html>

```

Analiza la arquitectura. El header resuelve identidad y navegación en un solo vistazo. El main contiene exactamente cuatro secciones, cada una con su id que coincide con el menú: la navegación por anclas funciona sin JavaScript. La jerarquía de encabezados es impecable: un h1 para el título principal, h2 para cada sección, h3 para cada proyecto y para "Mis habilidades". Los proyectos usan <article> porque son piezas autocontenidas. La sección de contacto ya integra el formulario profesional de esta unidad: fieldset con legend para los datos y textarea para el mensaje. El footer cierra con derechos, navegación secundaria y una firma con personalidad. Esta página, sin una sola línea de CSS, ya está bien estructurada; cuando aprendas CSS, solo tendras que vestirla.

Ejemplo 4: footer profesional reutilizable

El footer suele hacerse al final y con prisa, y se nota. Este ejemplo muestra un footer completo y bien pensado que puedes reutilizar en todos tus proyectos. Incluye cuatro bloques: marca, navegación, contacto y legal.

```
<footer>

  <!-- Bloque 1: identidad y frase de marca -->
  <div>
    
    <p>Desarrollador web en formacion. Codigo limpio, paginas claras.</p>
  </div>

  <!-- Bloque 2: navegacion secundaria para paginas largas -->
  <nav>
    <p><strong>Mapa del sitio</strong></p>
    <ul>
      <li><a href="#inicio">Inicio</a></li>
      <li><a href="#sobre-mi">Sobre mi</a></li>
      <li><a href="#proyectos">Proyectos</a></li>
      <li><a href="#contacto">Contacto</a></li>
    </ul>
  </nav>

  <!-- Bloque 3: como contactarte sin rodeos -->
  <div>
    <p><strong>Contacto</strong></p>
    <ul>
      <li>Correo: hola@estebanega.dev</li>
      <li>GitHub: github.com/estebanega</li>
      <li>LinkedIn: linkedin.com/in/estebanega</li>
    </ul>
  </div>

  <!-- Bloque 4: la linea legal que todo sitio serio tiene -->
  <div>
    <p>&copy; 2026 Esteban Ega. Todos los derechos reservados.</p>
    <p><a href="#aviso">Aviso de privacidad</a> | <a href="#terminos">Terminos de
uso</a></p>
  </div>

</footer>
```


Por que este footer funciona: cada bloque responde a una necesidad real del visitante que llevo hasta abajo de la página. El bloque de marca le recuerda quien eres. El mapa del sitio le evita subir hasta arriba para navegar. El contacto le da tus datos sin tener que buscarlos. Y la línea legal transmite seriedad: los sitios profesionales siempre la tienen. Nota el uso de **en los títulos de bloque: es importancia real, no solo negritas decorativas. Y los enlaces de "Aviso de privacidad" apuntan a anclas que crearas cuando esas páginas existan: es válido planear enlaces futuros siempre que los crees antes de publicar.**

Práctica

Ejercicio 1: formulario de inscripción a un taller

Enunciado: crea un formulario de inscripción a un taller de cocina que use al menos dos fieldset con sus legend. El primer grupo se llamara "Datos del participante" e incluire nombre y correo obligatorios. El segundo grupo se llamara "Detalles del taller" e incluire un campo de ciudad con datalist (sugiere al menos cuatro ciudades) y un select del taller deseado con optgroup que agrupe por categoría (por ejemplo, "Reposteria" y "Cocina salada", con dos opciones cada una). Agrega un boton de envio.

Solución explicada paso a paso:

Paso 1: abre la estructura base del documento (DOCTYPE, html, head con meta charset y title, body). Todo formulario vive dentro de un documento completo.

Paso 2: crea el <form> con action y method. Aunque el destino sea ficticio por ahora, acostumbra a declararlo: un form sin action es como una carta sin destinatario.

Paso 3: abre el primer <fieldset> y pon <legend>Datos del participante</legend> como primer hijo. Dentro, agrega las etiquetas label e input para nombre (type text, required) y correo (type email, required). El email con type email te regala validación automática.

Paso 4: abre el segundo <fieldset> con <legend>Detalles del taller</legend>. Dentro, crea el input de texto para ciudad con el atributo list apuntando a un id, por ejemplo list="ciudades", y después el <datalist id="ciudades"> con cuatro <option> con sus value.

Paso 5: agrega el <select> para el taller con una primera opción vacía que diga "-- Elige un taller --", y dos <optgroup> con su label cada uno, cada uno con dos <option>.

Paso 6: cierra los fieldset, agrega el <button type="submit"> y cierra el form y el documento. Abrelo en el navegador y prueba: escribe una ciudad que no este en la lista para comprobar que el datalist la acepta.

```
<form action="/inscripcion" method="post">
  <fieldset>
    <legend>Datos del participante</legend>
    <label for="nombre">Nombre completo:</label>
    <input type="text" id="nombre" name="nombre" required>
    <br><br>
    <label for="email">Correo electronico:</label>
    <input type="email" id="email" name="email" required>
  </fieldset>
  <br>
  <fieldset>
    <legend>Detalles del taller</legend>
    <label for="ciudad">Ciudad:</label>
    <input type="text" id="ciudad" name="ciudad" list="ciudades">
    <datalist id="ciudades">
      <option value="Ciudad de Mexico">
      <option value="Guadalajara">
      <option value="Monterrey">
      <option value="Oaxaca">
    </datalist>
    <br><br>
    <label for="taller">Taller deseado:</label>
    <select id="taller" name="taller" required>
      <option value="">-- Elige un taller --</option>
      <optgroup label="Reposteria">
        <option value="pasteles">Pasteles desde cero</option>
        <option value="pan">Pan artesanal</option>
      </optgroup>
      <optgroup label="Cocina salada">
        <option value="mexicana">Cocina mexicana</option>
        <option value="italiana">Cocina italiana</option>
      </optgroup>
    </select>
  </fieldset>
  <br>
  <button type="submit">Inscribirme al taller</button>
</form>
```

Ejercicio 2: caza de etiquetas obsoletas

Enunciado: el siguiente código lo copio un amigo de un tutorial del 2009. Encuentra las tres etiquetas obsoletas, explica por que cada una está mal y reescribe el fragmento de forma moderna.

```
<center>
  <font size="6" color="green">Bienvenidos a mi tienda</font>
</center>
<marquee>Ofertas todos los dias</marquee>
```

Solución explicada paso a paso:

Paso 1: identifica `<center>`. Es obsoleta porque centrar es presentación, no estructura. Su reemplazo moderno es CSS: `text-align: center` para centrar texto.

Paso 2: identifica `<font>` con sus atributos `size` y `color`. Es obsoleta porque mezcla diseño con contenido y es imposible de mantener a escala. Su reemplazo moderno es CSS: `font-size` y `color`.

Paso 3: identifica `<marquee>`. Es obsoleta por accesibilidad (el movimiento molesta y puede hacer dano), por falta de control del usuario y por mezclar comportamiento con estructura. Su reemplazo moderno es texto quieto y legible, y si se quiere animación, animaciones CSS controlables.

Paso 4: reescribe aplicando el criterio "HTML dice que es, CSS dice como se ve". El título es un encabezado, así que usa `h2`; el anuncio es un párrafo con énfasis real, así que usa `p` con `strong`.

```
<!-- Version moderna del código del amigo -->
<h2 style="text-align: center; font-size: 32px; color: green;">Bienvenidos a mi
tienda</h2>
<p><strong>Ofertas todos los dias</strong></p>
```

Lección extra: cuando encuentres código así en tutoriales, revisa la fecha de publicación. Si el tutorial es anterior a 2014 y enseña `<center>` o `<font>` como si nada, busca otra fuente. Tu criterio vale más que cualquier tutorial viejo.

Ejercicio 3: strong y em contra b e i, el examen del oído

Enunciado: decide en cada caso si la etiqueta correcta es , <i>, o , y justifica con la prueba del oído: "si alguien escuchara esta página, esta palabra debería sonar enfatizada?".

Caso A: el nombre científico de una planta en un artículo de botánica, que por convención va en cursiva: "La <...>Monstera deliciosa</...> es originaria de México." Caso B: una advertencia en un formulario de pago: "<...>No compartas tu contraseña</...>: nadie del banco te la va a pedir." Caso C: el título de un libro mencionado en una reseña, que por convención va en cursiva: "En <...>Cien años de soledad</...>, García Márquez..." Caso D: la palabra clave de una oferta que el negocio quiere enfatizar de verdad: "Descuento del <...>50%</...> solo esta semana."

Solución explicada paso a paso:

Paso 1: caso A. Es convención tipográfica (nombres científicos en cursiva), no énfasis real de lectura. Si lo escuchas, no necesitas entonación especial. Respuesta: <i>. Código: La <i>Monstera deliciosa</i> es originaria de México.

Paso 2: caso B. Es una advertencia de seguridad genuinamente importante: si la escuchas, debe sonar con alarma. Respuesta: . Código: No compartas tu contraseña: nadie del banco te la va a pedir.

Paso 3: caso C. Igual que el caso A: convención tipográfica para títulos de obras, sin énfasis de lectura. Respuesta: <i>. Código: En <i>Cien años de soledad</i>, García Márquez...

Paso 4: caso D. El negocio quiere que el 50 por ciento retumbe en la mente del lector: es énfasis real con intención comunicativa. Respuesta: . Código: Descuento del 50% solo esta semana.

La prueba del oído nunca falla: si la palabra debe sonar distinta al escucharse, usa strong o em; si solo debe verse distinta por convención visual, usa b o i. Los lectores de pantalla y los buscadores te lo agradecen.

Ejercicio 4: arma la estructura base de TU portafolio

Enunciado: usando el ejemplo 3 como referencia pero con tus propios datos, crea tu archivo index.html con: header con tu nombre como logo y nav con cuatro enlaces, main con secciones de inicio (con h1), sobre mí, proyectos (con al menos dos article) y contacto (con un formulario que use fieldset, legend y textarea), y footer con derechos reservados y tu firma.

Además, crea en tu computadora la estructura de carpetas: una carpeta para el proyecto con subcarpetas css, img, js y docs.

Solución explicada paso a paso:

Paso 1: en tu computadora, crea la carpeta del proyecto, por ejemplo "mi-portafolio", y dentro crea las subcarpetas css, img, js y docs. Hazlo antes de escribir código: primero el archivero, luego los papeles.

Paso 2: dentro de "mi-portafolio", crea el archivo index.html y escribe la estructura base: DOCTYPE, html con lang="es", head con meta charset, meta viewport y un title con tu nombre y tu rol.

Paso 3: escribe el <header>: un div con tu nombre (y si tienes un logo, la imagen con su alt) y un <nav> con una de cuatro enlaces a #inicio, #sobre-mi, #proyectos y #contacto.

Paso 4: escribe el <main> con las cuatro <section>, cada una con su id correspondiente. La de inicio lleva el <h1> con tu nombre y una frase que diga que haces. La de sobre mi lleva un h2, un párrafo honesto y una lista de habilidades. La de proyectos lleva un h2 y dos <article>, cada uno con h3, descripción y enlace. La de contacto lleva un h2 y el formulario: un fieldset con legend para nombre y correo, y un textarea para el mensaje.

Paso 5: escribe el <footer> con la línea de derechos usando ©, el año actual y tu nombre; agrega un nav secundario y una frase de firma con tu personalidad.

Paso 6: abre index.html en el navegador y haz clic en cada enlace del menú: cada uno debe llevarte a su sección. Si alguno no funciona, revisa que el id de la sección sea idéntico al del href, incluyendo guiones y minúsculas.

Paso 7: verifica tu trabajo con esta lista: un solo main, un solo h1, jerarquía de encabezados en orden, todos los id coinciden con los enlaces, el formulario tiene fieldset con legend, el textarea tiene etiqueta de cierre, y no hay ni una etiqueta obsoleta. Si todo está en verde, tienes los cimientos de tu portafolio.

Ejercicio 5: formulario de encuesta con select multiple y textarea

Enunciado: crea una encuesta de salida para un curso con: un fieldset llamado "Tu experiencia" que contenga un select multiple (con size 4) donde el estudiante elija los temas que más le gustaron (al menos cinco opciones), incluyendo una instrucción breve sobre como elegir varias; y un textarea para sugerencias de mejora, con rows 5, cols 50, maxlength 250 y un placeholder orientador. Todo dentro de un form con boton de envio.

Solución explicada paso a paso:

Paso 1: crea el form con su action y method, y dentro el `<fieldset>` con `<legend>Tu experiencia</legend>` como primer hijo.

Paso 2: agrega la etiqueta label para el select multiple. En el texto de la etiqueta incluye la instrucción: "(manten presionada la tecla Ctrl para elegir varias opciones)". Este detalle separa un formulario amateur de uno profesional.

Paso 3: crea el `<select>` con los atributos `multiple` y `size="4"`, un `id` y un `name`. Agrega al menos cinco `<option>`, cada una con su `value`.

Paso 4: agrega el label y el `<textarea>` para sugerencias: `id`, `name`, `rows="5"`, `cols="50"`, `maxlength="250"` y un placeholder como "Que mejorarias del curso? Se específico, por favor...". Recuerda pegar la etiqueta de cierre sin espacios entre la apertura y el cierre para no matar el placeholder.

Paso 5: agrega el boton de envio, cierra el `fieldset` y el `form`, y prueba en el navegador: elige varias opciones con Ctrl más clic y escribe más de 250 caracteres en el `textarea` para comprobar que el limite funciona.

```

<form action="/encuesta" method="post">
  <fieldset>
    <legend>Tu experiencia</legend>

    <label for="gustos">Que temas te gustaron mas? (manten Ctrl para elegir
varios)</label><br>
    <select id="gustos" name="gustos" multiple size="4">
      <option value="html">Estructura con HTML</option>
      <option value="formularios">Formularios</option>
      <option value="tablas">Tablas</option>
      <option value="multimedia">Audio y video</option>
      <option value="semantica">HTML semantico</option>
      <option value="proyecto">El proyecto del portafolio</option>
    </select>
    <br><br>

    <label for="mejoras">Sugerencias para mejorar el curso:</label><br>
    <textarea id="mejoras" name="mejoras" rows="5" cols="50" maxlength="250"
placeholder="Que mejorarias del curso? Se especifico, por favor..."></textarea>
  </fieldset>
  <br>
  <button type="submit">Enviar encuesta</button>
</form>

```

Resumen de la unidad

- fieldset agrupa campos relacionados de un formulario dentro de un recuadro visible, como carpetas en un archivero; legend le pone título al grupo y debe ser su primer hijo.
- datalist ofrece sugerencias a un campo de texto sin obligar al usuario: puede elegir una sugerencia o escribir libremente; se conecta al input mediante el atributo list y el id.
- optgroup organiza las opciones de un select en categorías con títulos no seleccionables mediante el atributo label; ideal cuando la lista es larga y tiene grupos naturales.
- El atributo multiple convierte un select en selección multiple, y size controla cuantas opciones se ven; agrega siempre una instrucción breve porque el mecanismo de Ctrl más clic no es obvio.
- textarea es el control para textos largos, con etiqueta de cierre obligatoria; rows y cols controlan el tamaño visible, y maxlength, placeholder y required le dan control fino.

- Las etiquetas <center>, y <marquee> estan obsoletas: fueron útiles en los noventa pero hoy su trabajo pertenece a CSS; usarlas delata código desactualizado.
- e <i> no estan obsoletas, pero solo aportan apariencia; y aportan significado real de importancia y énfasis que escuchan los lectores de pantalla y valoran los buscadores.
- La regla de oro: HTML es estructura, no estilo; si quieres cambiar apariencia, la respuesta está en CSS, no en buscar una etiqueta mágica.
- Tu portafolio nace en esta unidad: header con logo y navegación, un solo main con secciones identificadas con id, y footer con derechos reservados, navegación secundaria y firma.
- Organiza tu proyecto como un profesional: carpeta del proyecto con subcarpetas css, img, js y docs; nombres en minúsculas con guiones; rutas relativas siempre.
- La navegación por anclas conecta el menú con las secciones mediante id idénticos; sin esos id, el menú es decoración.
- Planifica antes de codificar: objetivo en una frase, secciones definidas y textos escritos; un buen plano construye el edificio en la mitad de tiempo.

UNIDAD 10: Proyecto final y graduación

Llegaste. Después de nueve unidades, veintisiete clases y cientos de etiquetas practicadas, aquí estás, en la última unidad del curso. Y te voy a decir algo desde este primer párrafo, sin rodeos: esta unidad no se trata de aprender más etiquetas. Se trata de demostrarte a ti mismo que ya sabes usar las que aprendiste.

Piensa en el camino que recorriste para llegar aquí. En la unidad 1 abriste tu primer archivo .html, escribiste unas líneas en un editor de texto plano y el navegador te obedeció mostrándolas en pantalla. ¿Te acuerdas de esa sensación? Parecía magia. Era la primera vez que le dabas instrucciones directas a un navegador y él las seguía al pie de la letra. Ese día dejaste de ser un simple usuario de internet y te convertiste en alguien que construye internet. No fue un día cualquiera, aunque en ese momento no lo supieras.

Después vinieron los encabezados y los párrafos, las listas y los enlaces, las imágenes y las tablas. Aprendiste a pedirle datos al visitante con formularios, a organizar el contenido con etiquetas semánticas, a meter audio y video en tus páginas y a validar lo que el usuario escribe sin necesidad de programar nada más. Cada unidad te entregó una herramienta nueva y cada ejercicio te enseñó a sostenerla con más seguridad. Si las primeras nueve unidades fueron tu caja de herramientas, esta última unidad es el taller donde por fin vas a construir algo completo con ellas.

Y eso es exactamente lo que cubre esta unidad. La clase 28 te enseña a darle forma a tu portafolio con section, article y aside: la sección "sobre mí" que cuenta quién eres, el artículo de tu blog donde compartes lo que aprendes y la barra lateral con links útiles. La clase 29 te enseña a construir el formulario de contacto final, con validación real, para que cualquier persona que visite tu portafolio pueda escribirte. Y la clase 30 es la graduación: el repaso de todo lo logrado y el mapa de lo que sigue en tu camino como desarrollador frontend.

¿Por qué importa tanto terminar con un proyecto y no con un examen? Porque en el mundo real nadie te contrata por recitar etiquetas de memoria. Te contratan, te recomiendan y te pagan por lo que eres capaz de construir. Un cocinero no demuestra lo que sabe describiendo ingredientes: cocina un platillo y te lo sirve. Un mecánico no te convence hablando de herramientas: arregla tu coche y te lo entrega funcionando. Tu portafolio es ese platillo servido, ese coche arreglado. Es la prueba tangible de que no solo estudiaste HTML5, sino que lo dominas lo suficiente para crear algo útil, ordenado y publicable.

Fíjate cómo encaja esta unidad en el curso completo. Las unidades 1 a 3 te enseñaron el lenguaje básico: la estructura del documento, el texto, las listas, los enlaces y las imágenes.

Esos eran los ladrillos. Las unidades 4 y 5 te enseñaron a interactuar con el visitante a través de tablas y formularios. Las unidades 6 a 8 te enseñaron a organizar y enriquecer: semántica, multimedia, buenas prácticas. La unidad 9 te enseñó los detalles finos: SEO básico, navegación interna y validación. Cada unidad fue un piso de la construcción. Esta unidad 10 es el techo y la entrega de llaves: tomas todo lo anterior y lo ensamblas en una sola página que puedes publicar, compartir y presumir.

Hablemos de lo que vas a construir, pieza por pieza, para que no sea una caja negra. Tu portafolio tendrá una sección "sobre mí" donde te presentas como profesional: quién eres, qué sabes hacer y qué te apasiona. Tendrá una sección de proyectos donde muestras lo que has construido durante el curso. Tendrá un blog con tus primeros artículos, porque escribir sobre lo que aprendes es la forma más rápida de dominarlo. Y tendrá un formulario de contacto para que reclutadores, clientes o colegas puedan escribirte. Todo conectado con un menú de navegación que salta de una sección a otra sin recargar la página. Cuando termines, no tendrás "un ejercicio más": tendrás tu casa digital.

Quiero advertirte algo que le pasa a casi todos los estudiantes en este punto: el síndrome del impostor. Vas a mirar tu portafolio terminado y una vocecita te va a decir que "solo es HTML", que "todavía no sabes CSS", que "cualquiera lo hace". Esa vocecita miente. Hace nueve unidades no sabías qué era una etiqueta y hoy vas a publicar una página completa, semántica, con navegación interna, blog y formulario validado. Eso no lo hace "cualquiera": lo hace alguien que estudió, practicó y terminó. El impostor es el que finge saber; tú tienes el archivo para demostrarlo.

Una recomendación antes de empezar, y tómatela en serio: en esta unidad está prohibido copiar y pegar sin entender. Vas a ver ejemplos completos y soluciones detalladas, pero tu trabajo es escribir cada línea con tus propias manos, aunque te tardes más. El que copia un portafolio tiene un archivo bonito y las manos vacías; el que lo escribe tiene un archivo bonito y la habilidad para hacer diez más. Cuando algo no te salga, no borres todo: lee con calma, revisa la etiqueta, compara con el ejemplo. Equivocarte aquí, en privado, es gratis, y es la forma más rápida de aprender.

Al final de esta unidad vas a tener tres cosas. Primera: un archivo `index.html` completo con tu portafolio, listo para subirse a internet. Segunda: una lista de verificación para saber, con criterios claros, que está bien hecho. Tercera: el mapa claro de tu siguiente paso, porque HTML5 fue solo el inicio del camino. ¿Listo? Vamos a construir.

Explicación detallada

Tu proyecto final: un portafolio de verdad

Un portafolio es tu escaparate profesional. Piensa en una tienda de la esquina: el dueño no guarda sus mejores productos en la bodega, los pone en el aparador, bien iluminados, para que quien pase por la banqueta se detenga a mirarlos. Tu portafolio es ese aparador. No es un currículum en PDF que nadie abre; es una página viva que demuestra, con hechos, lo que sabes hacer.

¿Qué lleva un buen portafolio de alguien que va empezando? Cuatro cosas, ni una más. Primero, quién eres: una sección "sobre mí" con tu foto, una descripción honesta y tus habilidades. Segundo, qué has hecho: una sección de proyectos con lo que construiste en el curso. Tercero, qué piensas: un blog con artículos donde explicas lo que aprendes, porque enseñar es la prueba máxima de que entendiste. Cuarto, cómo contactarte: un formulario para que te escriban. Con esas cuatro piezas ya tienes algo que ningún principiante improvisado tiene: presencia profesional en internet.

Y aquí viene lo bonito: ya sabes construir las cuatro. No hay ni una sola etiqueta en tu portafolio que no hayas practicado en las unidades anteriores. La clase 28 te ordena las piezas con `section`, `article` y `aside`, y la clase 29 te da el formulario. Esta unidad no te pide saltar más alto; te pide usar todo lo que ya sabes saltar, pero ahora con puntería.

section con id: cada parte con nombre y apellido

La etiqueta `section` agrupa contenido que forma una unidad temática dentro de tu página. Si tu portafolio fuera un libro, cada `section` sería un capítulo: "sobre mí" es un capítulo, "proyectos" es otro, "blog" es otro, "contacto" es otro. Cada capítulo tiene su tema, su inicio y su fin.

Pero un capítulo sin título es difícil de encontrar. Para eso sirve el atributo `id`: le pone nombre y apellido a cada sección. Cuando escribes `section id="sobre-mi"`, le estás diciendo al navegador: "esta sección se llama sobre-mi, y no hay otra igual en todo el documento". El `id` es único, como tu CURP: puede haber muchas personas, pero cada CURP identifica a una sola.

Mira la estructura básica:

```
<section id="sobre-mi">
  <h2>Sobre mí</h2>
  <p>Hola, soy desarrollador web en formación...</p>
</section>
```

El id tiene reglas que debes respetar siempre. Primera: no lleva espacios. "sobre mi" con espacio es inválido; usa guiones: "sobre-mi". Segunda: no uses tildes ni eñes en los ids, aunque en el texto visible sí puedes. El id es una llave técnica, mantenlo simple: letras minúsculas, números y guiones. Tercera: no repitas el mismo id dos veces en la página. Si dos secciones se llaman igual, el navegador se confunde y tu navegación por anclas fallará. Cuarta: que el id describa el contenido. id="seccion1" no dice nada; id="proyectos" lo dice todo.

Un error muy común en proyectos reales es poner ids genéricos o decorativos que después nadie usa. Cada id que pongas debe tener un propósito: o sirve para la navegación por anclas, o servirá después para darle estilo con CSS o para manipularlo con JavaScript. Un id sin propósito es como ponerle nombre a una caja vacía.

article: tu blog dentro del portafolio

Si section es el capítulo del libro, article es el cuento que puedes sacar del libro y leer por separado sin perder el sentido. La etiqueta article envuelve contenido independiente y autocontenido: un post de tu blog, una noticia, una tarjeta de proyecto. La prueba de fuego es esta: si puedes tomar ese bloque, publicarlo solo en otra página y sigue teniendo sentido completo, entonces es un article.

Tu blog dentro del portafolio funciona así. Cada entrada es un article con su título, su fecha y su contenido:

```
<article>
  <h2>Lo que aprendí construyendo mi primer formulario</h2>
  <p>Publicado el 19 de septiembre de 2026</p>
  <p>Al principio pensé que un formulario era solo cajas de texto...</p>
</article>
```

¿Por qué un blog dentro del portafolio de un principiante? Por tres razones muy prácticas. Primera: escribir te obliga a ordenar tus ideas, y lo que puedes explicar con claridad es lo que realmente dominas. Segunda: un reclutador que lee tus artículos ve cómo piensas, no solo qué etiquetas conoces. Tercera: te da contenido propio para compartir y una razón para volver a tu portafolio cada semana a publicar algo nuevo. Un portafolio abandonado dice "aprendí y me detuve"; un blog activo dice "aprendo todos los días".

La diferencia entre section y article confunde a muchos, así que grábate esta analogía. Imagina un periódico. El periódico completo es tu página. Cada sección del periódico (deportes, cultura, tecnología) es un section: agrupa contenido por tema. Cada nota individual

dentro de la sección de deportes es un article: tiene su titular, su autor y se entiende sola. En tu portafolio, "blog" es el section y cada post es un article. ¿Ves? No compiten, se complementan: el section organiza, el article contiene.

Otro error común: envolver todo en article "porque se ve más semántico". No. Si el contenido no se sostiene solo, no es article. Tu sección "sobre mí" no es un article, es un section. La lista de tus habilidades no es un article. Reserva article para piezas independientes: posts, proyectos individuales, testimonios.

aside: links útiles al margen

La etiqueta aside es para el contenido relacionado pero secundario: lo que enriquece sin ser lo principal. Piensa en un libro de texto: el contenido central va en las páginas, y al margen hay recuadros con datos curiosos, definiciones o lecturas recomendadas. Eso es aside: la nota al margen.

En tu portafolio, el aside brilla en dos lugares. Primero, dentro de un article: al final de tu post del blog puedes poner un aside con "links útiles" o "para seguir aprendiendo". Segundo, a nivel de página: una barra lateral con tus redes sociales, recursos favoritos o tecnologías que estás aprendiendo.

```
<article>
  <h2>Mi primer post</h2>
  <p>Contenido del post...</p>
  <aside>
    <h3>Links útiles</h3>
    <ul>
      <li><a href="https://lucasegaacademy.com">Lucas EGA Academy</a></li>
      <li><a href="https://developer.mozilla.org">MDN Web Docs</a></li>
    </ul>
  </aside>
</article>
```

Fíjate en la posición: el aside vive dentro del article porque esos links complementan ese post específico. Si los links fueran generales de todo tu sitio, el aside iría fuera, como hermano de las secciones. La regla es simple: aside complementa a su ancestro más cercano. Dentro de un article, complementa al article; directo en el body o en el main, complementa a la página.

El error clásico con aside es usarlo como basurero: meter ahí todo lo que no supiste dónde poner. Si el contenido no tiene relación clara con lo que complementa, no va en un aside, va en su propia sección o no va. Un aside con links rotos o irrelevantes resta más de lo que suma.

Navegación por anclas: el mapa de tu portafolio

Ya tienes secciones con nombre. Ahora necesitas un mapa para moverte entre ellas sin perderte. Ese mapa es la navegación por anclas: enlaces que saltan a una parte específica de la misma página usando el id como destino.

Funciona así. En tu menú escribes un enlace cuyo href empieza con el símbolo de gato seguido del id:

```
<nav>
  <ul>
    <li><a href="#sobre-mi">Sobre mí</a></li>
    <li><a href="#proyectos">Proyectos</a></li>
    <li><a href="#blog">Blog</a></li>
    <li><a href="#contacto">Contacto</a></li>
  </ul>
</nav>
```

Cuando el visitante hace clic en "Proyectos", el navegador busca en la página el elemento con id="proyectos" y se desplaza hasta él, sin recargar nada. Es como un edificio con elevador: cada botón del elevador te lleva directo al piso que quieres, sin pasar por las escaleras. El id es el número de piso y el enlace con gato es el botón.

Esta técnica la aprendiste en la unidad 9 y aquí se vuelve protagonista, porque es lo que convierte cuatro secciones sueltas en un sitio navegable. Un portafolio sin navegación por anclas obliga al visitante a hacer scroll a ciegas buscando tu contacto; uno con navegación le pone todo a un clic de distancia. Los reclutadores revisan decenas de portafolios al día: el que les facilita la vida es el que recuerdan.

Tres detalles finos que separan al amateur del profesional. Primero: el texto del enlace debe coincidir con el título de la sección. Si el enlace dice "Sobre mí" pero la sección se titula "Quién soy", el visitante duda. Segundo: el orden del menú debe ser el mismo orden de las secciones en la página. Tercero: incluye siempre un enlace de regreso arriba o asegúrate de que el menú esté visible; nadie quiere quedarse atrapado al final de tu página sin saber cómo volver.

Errores comunes con secciones y anclas (y cómo evitarlos)

En proyectos reales, estos son los errores que más se repiten con sections, ids y anclas. Revisalos como si fuera una lista de seguridad antes de despegar.

Primero, ids duplicados. Copiaste la sección de proyecto para hacer la segunda y se te olvidó cambiar el id. Resultado: los dos enlaces del menú llevan al mismo lugar y la segunda sección es inalcanzable. Solución: cada vez que dupliques un bloque, cambia el id antes de seguir escribiendo.

Segundo, el enlace apunta a un id que no existe. Escribiste href="#contactos" en plural pero la sección es id="contacto" en singular. El clic no hace nada y el visitante piensa que tu página está rota. Solución: después de escribir el menú, haz clic en cada enlace y verifica que salta al lugar correcto.

Tercero, espacios o mayúsculas en los ids. id="Sobre Mi" parece inofensivo, pero los ids distinguen mayúsculas de minúsculas y los espacios los rompen. href="#sobre-mi" no encontrará a id="Sobre Mi". Solución: todo en minúsculas, palabras separadas con guiones, siempre.

Cuarto, secciones sin encabezado. Un section sin h2 es como un capítulo sin título: el lector no sabe dónde empieza un tema y termina otro. Cada section de tu portafolio debe abrir con su h2 correspondiente.

Quinto, abusar de section para todo. No cada bloque necesita ser section. Un grupo de botones, un banner decorativo o un texto suelto pueden vivir en un div o directamente en el main. La semántica es poderosa cuando se usa con criterio, no cuando se reparte como confeti.

El formulario de contacto: tu buzón profesional

Todo portafolio necesita una puerta de entrada para las oportunidades, y esa puerta es el formulario de contacto. Piensa en el form como un buzón con dirección: el visitante escribe su mensaje, lo deposita, y el buzón sabe exactamente a dónde llevarlo. Esa "dirección" es el atributo action.

```
<form action="/enviar" method="post">
  <!-- aquí van los campos -->
</form>
```

El atributo `action` dice a dónde se envían los datos cuando el visitante presiona el botón de enviar. En tu caso, `action="/enviar"` significa: "cuando alguien envíe este formulario, lleva los datos a la ruta `/enviar` del servidor". Hoy, con solo HTML, ese servidor aún no existe; lo construirás cuando aprendas backend o usarás un servicio que reciba los datos por ti. Y está bien: tu trabajo ahora es que el buzón esté bien construido, con ranura del tamaño correcto y cerradura funcionando. El cartero (el backend) llegará después.

El atributo `method` indica cómo viajan los datos. Con `method="post"` los datos viajan ocultos, dentro del envío; con `method="get"` viajarían visibles en la dirección web. Para un formulario de contacto, `post` es lo correcto: nadie quiere que su mensaje personal aparezca en la barra de direcciones del navegador. Grábate la regla: si el formulario envía información sensible o mensajes, usa `post`; si es un buscador donde quieres compartir la liga con los resultados, usa `get`.

Dentro del `form` viven los campos. Para un formulario de contacto profesional necesitas tres: nombre, correo y mensaje. Cada campo lleva su label conectado con `for`, su `id` único y su `name`, que es el nombre con el que viajará el dato al servidor. Mira el campo de correo, que es el corazón del formulario:

```
<label for="email">Correo electrónico:</label>
<input type="email" id="email" name="email" required>
```

Tres atributos trabajan en equipo aquí. El `type="email"` le dice al navegador: "esto debe parecer un correo". El `required` le dice: "sin esto no se envía nada". El `name="email"` le dice al servidor: "este dato se llama email". Quita cualquiera de los tres y algo se rompe: sin `type`, acepta cualquier texto; sin `required`, se envía vacío; sin `name`, el dato se pierde en el camino aunque el usuario lo haya escrito. En proyectos reales, el error más caro es el `name` olvidado: el formulario "funciona", el usuario lo envía feliz, y al servidor no le llega nada.

El campo de mensaje usa `textarea`, que aprendiste en la unidad 5. A diferencia del `input`, que es de una sola línea, el `textarea` es un cuadro de texto amplio donde el visitante puede expresarse:

```
<label for="mensaje">Mensaje:</label>
<textarea id="mensaje" name="mensaje" rows="5" cols="40" required minlength="10"
placeholder="Cuéntame en qué puedo ayudarte..."></textarea>
```

Los atributos `rows` y `cols` definen el tamaño visible: 5 renglones de alto por 40 columnas de ancho, un tamaño cómodo para escribir un mensaje. El `placeholder` muestra un texto de ayuda

que desaparece cuando el usuario empieza a escribir. Y `minlength="10"` exige un mínimo de 10 caracteres: adiós a los mensajes de "hola" sin contenido. Fíjate que el `textarea` no se cierra solo: siempre lleva etiqueta de apertura y de cierre, con el contenido inicial (si lo hay) entre ellas.

Y para cerrar, el botón que lo activa todo:

```
<button type="submit">Enviar mensaje</button>
```

El `type="submit"` es lo que convierte un botón bonito en el disparador del envío. Un `button` sin `type` dentro de un `form` se comporta como `submit` por defecto, pero escribirlo explícitamente es una buena práctica: deja clara tu intención y evita sorpresas cuando agregues otros botones después. El texto del botón también importa: "Enviar mensaje" es claro y humano; "Submit" es frío y genérico. Tu portafolio habla de ti hasta en los botones.

Validación aplicada: deja que el navegador trabaje por ti

La validación es el guardia de seguridad de tu formulario: revisa que lo que el visitante escribe tenga sentido antes de dejarlo pasar. Y la gran noticia es que HTML5 trae ese guardia incluido, gratis, sin programar nada. Lo aprendiste en la unidad 9; aquí lo aplicas en serio.

El atributo `required` es el guardia más estricto: el campo no puede irse vacío. El navegador bloquea el envío y muestra un mensaje automático: "Completa este campo". El atributo `type="email"` es el guardia especializado: verifica que el texto tenga formato de correo, es decir, algo antes del arroba, el arroba, y algo después con un punto. Si alguien escribe "juan@" o "juan.com", el navegador lo detiene y avisa.

Puedes afinar más. Con `minlength` y `maxlength` controlas el largo: `minlength="10"` en el mensaje evita textos vacíos de contenido, `maxlength="500"` evita novelas. Con `pattern` puedes exigir formatos específicos usando expresiones regulares, por ejemplo que un teléfono tenga cierto formato, aunque para tu formulario de contacto con `required`, `type="email"`, `minlength` y `maxlength` tienes más que suficiente.

Hay un detalle que muchos pasan por alto: la validación de HTML5 ocurre en el navegador, del lado del cliente. Es rápida y mejora la experiencia, pero no es una muralla de seguridad: alguien con conocimientos podría saltársela. Por eso los profesionales validan dos veces: en el navegador (para ayudar al usuario) y en el servidor (para proteger los datos). Tú hoy dominas la primera; la segunda llegará con el backend. No te angusties: para tu portafolio, la validación del navegador es exactamente lo que necesitas.

Un consejo de oro sobre los mensajes de error: no pelees con los mensajes automáticos del navegador al inicio. Están traducidos al idioma del usuario, son claros y funcionan. Ya habrá tiempo, con JavaScript, de personalizarlos. Hoy tu energía debe ir en que la validación exista y funcione, no en que se vea bonita.

El mensaje de confirmación: cierra el ciclo con el visitante

Imagina que depositas una carta en un buzón y el buzón no hace ningún ruido, no se mueve nada, no pasa nada. ¿Se envió? ¿Se perdió? Esa incertidumbre es horrible, y es exactamente lo que siente un visitante cuando envía tu formulario y la página no le confirma nada. El mensaje de confirmación cierra el ciclo: le dice "tu mensaje llegó, gracias, te responderé pronto".

Con solo HTML, la confirmación más honesta es una segunda página: gracias.html. El flujo es así: el visitante llena el formulario, presiona enviar, los datos viajan al action, y el servidor lo redirige a una página que dice "¡Mensaje recibido!". Esa página lleva tu agradecimiento, un estimado de respuesta ("te respondo en menos de 48 horas") y un enlace para volver al portafolio.

```
<!-- gracias.html -->
<h1>¡Gracias por escribirme!</h1>
<p>Tu mensaje llegó correctamente. Te responderé en menos de 48 horas.</p>
<a href="index.html">Volver al portafolio</a>
```

Mientras no tengas backend, hay un truco honesto y muy usado: deja el mensaje de confirmación visible junto al formulario, como una promesa de lo que pasará. Un párrafo debajo del form que diga "Al enviar, recibirás una confirmación y te responderé en menos de 48 horas" ya comunica profesionalismo. Lo importante es que el visitante nunca se quede con la duda.

Cuando aprendas JavaScript, podrás mostrar la confirmación sin cambiar de página: un mensaje que aparece debajo del formulario al enviarlo. Y con backend real, podrás guardar el mensaje y hasta enviar un correo automático de respuesta. Cada capa que aprendas mejora este ciclo, pero la base que pones hoy (el form bien construido y la confirmación contemplada) es la que sostiene todo lo demás.

Errores comunes en formularios reales

Los formularios son donde más se nota la diferencia entre un estudiante y un profesional, porque los errores aquí los sufre directamente el visitante. Estos son los que veo una y otra vez en proyectos reales.

Primero, labels desconectados. El label dice "Correo:" pero le falta el for, o el for apunta a un id que no existe. El daño es silencioso pero real: el usuario ya no puede hacer clic en el texto "Correo:" para activar el campo, y los lectores de pantalla no saben qué campo es cuál. Solución: cada input lleva su id único y cada label lleva su for apuntando a ese id. Sin excepciones.

Segundo, inputs sin name. El formulario se ve perfecto, válida perfecto, se envía perfecto... y al servidor no le llega nada, porque los datos viajan con el name como etiqueta. Es como mandar una carta sin remitente ni destinatario: el cartero no sabe qué hacer con ella. Solución: todo input, textarea y select dentro de un form lleva name.

Tercero, el botón traidor. Pones un segundo botón, por ejemplo "Limpiar", sin type, y resulta que por defecto actúa como submit y envía el formulario a la mitad. El usuario quería borrar y terminó enviando datos incompletos. Solución: todo button lleva su type explícito: submit para enviar, reset para limpiar, button para los que controlará JavaScript.

Cuarto, required en todo, incluso en lo opcional. Si marcas el teléfono como obligatorio cuando no lo necesitas, pierdes mensajes de gente que sí quería escribirte pero no quería darte su número. Solución: required solo en lo esencial (nombre, correo, mensaje). Cada campo obligatorio es una pequeña barrera; pon solo las necesarias.

Quinto, el form sin action ni method. Funciona en tu computadora porque "se ve bien", pero al publicarlo no hace nada útil. Solución: define el action desde el día uno, aunque apunte a una ruta que construirás después, y documenta con un comentario qué hará ese endpoint. Tu yo del futuro te lo agradecerá.

Repaso integral: todo lo aprendido, unidades 1 a 5

Antes de graduarte, vamos a hacer el recorrido completo. No es nostalgia: es inventario. Un profesional sabe exactamente qué herramientas tiene y para qué sirve cada una. Vamos unidad por unidad, y en cada una te voy a decir dónde vive ese conocimiento dentro de tu portafolio final.

Unidad 1: tus primeros pasos. Clases 1, 2 y 3. Aprendiste qué es HTML5 y por qué sigue siendo el lenguaje de la web después de tantos años: porque es el esqueleto de todo lo que

ves en internet. Aprendiste la anatomía del documento: el DOCTYPE que declara "esto es HTML5", la etiqueta html que envuelve todo, el head con la información invisible pero vital, y el body con todo lo que el visitante ve. Y creaste tu primera página y la abriste en el navegador. En tu portafolio, esta unidad vive en la estructura completa: cada vez que escribes el DOCTYPE y organizas head y body, estás usando la clase 2. Sin esta base, nada de lo demás se sostiene. Es como los cimientos de la casa: nadie los presume, pero sin ellos no hay casa.

Unidad 2: el texto manda. Clases 4, 5 y 6. Aprendiste los encabezados h1 a h6 y su jerarquía: un solo h1 por página como título principal, h2 para las grandes secciones, h3 para las subdivisiones. Aprendiste el párrafo con p, y el formato de texto: strong para lo importante, em para el énfasis, y las etiquetas de formato fino. En tu portafolio, esta unidad está en todas partes: el h1 con tu nombre en el encabezado, los h2 que titulan cada sección ("Sobre mí", "Proyectos", "Blog", "Contacto"), los párrafos donde te presentas y describes tus proyectos. El error que ya no cometes: usar h1 cinco veces porque "se ve grande". Hoy sabes que los encabezados son jerarquía, no tamaño de letra.

Unidad 3: listas, enlaces e imágenes. Clases 7, 8 y 9. Aprendiste las listas ordenadas (ol) para pasos y rankings, las desordenadas (ul) para conjuntos sin orden, y las de definición para pares de término y descripción. Aprendiste la etiqueta a con su href, la diferencia entre rutas absolutas y relativas, y los enlaces que abren en pestaña nueva. Aprendiste img con src y alt, y por qué el alt no es opcional: es la descripción para quien no puede ver la imagen y para Google. En tu portafolio, esta unidad vive en tu menú de navegación (una ul con enlaces), en tu lista de habilidades (ul), en tu foto de perfil (img con un alt que te describe) y en cada enlace de tu aside de links útiles. La clase 8, la de enlaces, es además la base de tu navegación por anclas de la clase 28: mismo mecanismo, nuevo truco.

Unidad 4: tablas y tus primeros formularios. Clases 10, 11 y 12. Aprendiste las tablas con table, tr, td y th: filas, celdas y encabezados para mostrar datos ordenados. Aprendiste tu primer form con su action, y los inputs básicos: text, password, number, y los atributos que los controlan como placeholder y value. En tu portafolio, la tabla puede vivir en tu sección de proyectos si quieres mostrar una comparativa de tecnologías, o en tu blog para datos ordenados. Pero lo más importante de esta unidad es el form: fue tu primer buzón, y el formulario de contacto de la clase 29 es su versión profesional y pulida. Todo lo que sabes de formularios empezó aquí, con un form sencillo y un botón.

Unidad 5: formularios que sí funcionan. Clases 13, 14 y 15. Aprendiste el label y cómo conectarlo con for para que cada campo tenga nombre y sea accesible. Aprendiste select con sus option para menús desplegables, textarea para textos largos, y fieldset con legend para agrupar campos relacionados. Aprendiste los botones: submit, reset y button, y cuándo usar

cada uno. En tu portafolio, esta unidad es el acabado fino del formulario de contacto: los labels conectados, el textarea del mensaje con sus rows y su placeholder, el button type="submit" con texto humano. Si la unidad 4 te dio el buzón, la unidad 5 te dio la ranura del tamaño exacto, la cerradura y el letrero de "depósito aquí".

Repaso integral: todo lo aprendido, unidades 6 a 10

Unidad 6: HTML semántico. Clases 16, 17 y 18. Esta fue la unidad que te convirtió de alguien que "pone etiquetas" en alguien que "estructura documentos". Aprendiste header para el encabezado, nav para la navegación, main para el contenido principal y footer para el pie. Aprendiste section y article para organizar el contenido por temas y piezas independientes, y aside para lo complementario. En tu portafolio, esta unidad ES el portafolio: el header con tu nombre y tu menú, el nav con las anclas, el main que envuelve tus secciones, el footer con tu copyright y tus redes. La clase 28 de esta unidad 10 es la aplicación directa de la clase 17 y 18: section con id para "sobre mí", article para tu post, aside para tus links. Si la unidad 6 te dio el vocabulario, la clase 28 te puso a escribir el discurso.

Unidad 7: contenedores y multimedia. Clases 19, 20 y 21. Aprendiste div y span, los comodines sin significado semántico pero utilísimos para agrupar y para lo que después estilizarás con CSS. Aprendiste a incrustar audio con la etiqueta audio y video con la etiqueta video, con sus controles y sus formatos. En tu portafolio, el div te sirve para agrupar bloques dentro de una sección cuando ningún elemento semántico aplica, y el span para marcar pedacitos de texto. El audio y el video pueden vivir en tu blog: imagina un post donde incrustas un video explicando tu proyecto, o tu sección de proyectos con un demo en video. No todos los portafolios los usan, pero el que sabe usarlos tiene una carta extra bajo la manga.

Unidad 8: detalles que separan al amateur del profesional. Clases 22, 23 y 24. Aprendiste los iframes para incrustar contenido externo como mapas o videos de otras plataformas. Aprendiste los atributos globales: id, class, title y lang, los que puedes poner en casi cualquier etiqueta. Y aprendiste los comentarios y las buenas prácticas del código limpio: indentación consistente, etiquetas en minúsculas, cerrar todo lo que abres. En tu portafolio, esta unidad vive en los detalles invisibles: los ids únicos y descriptivos de tus secciones, los comentarios que marcan dónde empieza y termina cada sección para que tu yo del futuro (o un reclutador que vea tu código) entienda tu organización, y el lang="es" en tu html que le dice al mundo que tu página está en español. Los profesionales se reconocen en estos detalles.

Unidad 9: navegación, validación y SEO básico. Clases 25, 26 y 27. Aprendiste los meta tags: charset para los acentos, viewport para los celulares, description para Google. Aprendiste las anclas y la navegación interna: los enlaces con gato que saltan a un id. Aprendiste la

validación de formularios con `required`, `type="email"`, `minlength` y los mensajes automáticos del navegador. En tu portafolio, esta unidad está en el `head` (esos metas que hacen que tu página se vea bien en el celular y aparezca decente en Google), en tu menú de navegación por anclas (la clase 26 convertida en protagonista) y en tu formulario de contacto validado (la clase 27 aplicada en la clase 29). Esta unidad fue el "casi listo": te dejó a un paso del proyecto final.

Unidad 10: proyecto final y graduación. Clases 28, 29 y 30. La clase 28 te dio las secciones del portafolio: `section` con `id="sobre-mi"`, `article` para tu primer post y `aside` para tus links útiles. La clase 29 te dio el formulario de contacto final: `form` con `action`, `input type="email"` `required`, `textarea` para el mensaje y `button` de envío, con validación aplicada y mensaje de confirmación. La clase 30 es esta celebración: el reconocimiento de que ya dominas HTML5 y el mapa hacia CSS3, JavaScript y la ruta frontend completa. Treinta clases. Diez unidades. Un portafolio. Cero excusas.

Si cuentas, son más de 60 etiquetas y atributos distintos los que pasaron por tus manos. No necesitas recitarlos de memoria: necesitas saber que existen y dónde buscarlos. El profesional no es el que se sabe todo de memoria; es el que sabe qué herramienta usar para cada problema y tiene el hábito de verificar su trabajo. Eso ya lo tienes.

Analogías para no olvidar nunca lo aprendido

Las analogías son atajos de memoria: cuando la teoría se borra, la imagen se queda. Estas son las que te acompañarán toda tu carrera.

HTML es el esqueleto del cuerpo. Sin esqueleto no hay forma, solo una masa informe. CSS será la piel, la ropa y el maquillaje. JavaScript será el sistema nervioso y los músculos: lo que se mueve y reacciona. Cuando alguien te pregunte qué hace cada tecnología, responde con el cuerpo humano y nunca te equivocarás.

Las etiquetas son cajas. Algunas cajas son grandes y ocupan todo el ancho (las de bloque, como `section` y `p`); otras son pequeñas y se sientan junto a otras (las de línea, como `span` y `a`). Armas tu página metiendo cajas dentro de cajas, como una mudanza bien organizada. El que entiende las cajas entiende el 80 por ciento del HTML.

El `id` es tu CURP y la `class` será tu uniforme. El CURP identifica a una sola persona en todo el país: así es el `id`, único en todo el documento. El uniforme lo usan todos los del equipo: así será la `class`, compartida por varios elementos. Nunca confundas los dos y tu código (y tu CSS futuro) te lo agradecerá.

El form es un buzón con dirección. El action es la dirección escrita en el buzón, el method es el servicio de paquetería (post discreto, get con la etiqueta visible), los inputs son las ranuras por donde entra cada dato, y el name es la etiqueta que identifica cada carta. Sin dirección, el buzón es decoración.

La validación es el guardia de la entrada. El required revisa que nadie entre sin identificación, el type="email" revisa que la identificación parezca real, el minlength revisa que el mensaje no sea un papel en blanco. Y como todo guardia amable, avisa antes de negar la entrada: esos son los mensajes automáticos del navegador.

Tu portafolio es tu aparador, no tu bodega. En la bodega guardas todo, hasta los ejercicios a medias. En el aparador pones solo lo mejor, bien iluminado. Cuando dudes si algo va en tu portafolio, pregúntate: "¿esto lo pondría en el aparador de mi tienda?" Si la respuesta es no, se queda en la bodega (tu carpeta de prácticas) y no pasa nada.

Checklist final: tu portafolio listo para publicarse

Antes de publicar, todo profesional pasa su trabajo por una lista de verificación. Esta es la tuya. Léela con tu archivo abierto y marca cada punto de verdad, no de memoria.

- El archivo se llama index.html y está en una carpeta dedicada a tu portafolio.
- Empieza con DOCTYPE html y la etiqueta html lleva lang="es".
- El head tiene meta charset="UTF-8", meta viewport y un title con tu nombre.
- Hay un solo h1 en toda la página, con tu nombre o tu título profesional.
- El header contiene tu presentación breve y el nav con la lista de enlaces.
- Cada enlace del menú apunta con gato a un id que sí existe en la página.
- Existe section id="sobre-mi" con h2, tu foto (con alt descriptivo), tu descripción y tu lista de habilidades.
- Existe section id="proyectos" con al menos dos proyectos descritos, aunque sean ejercicios del curso.
- Existe section id="blog" con al menos un article completo: título, fecha y contenido real.
- El article incluye un aside con links útiles relacionados con el post.
- Existe section id="contacto" con un form que tiene action y method="post".
- Cada campo del formulario tiene label con for, id único y name.
- El correo usa type="email" y los campos esenciales llevan required.

- El botón de envío es `button type="submit"` con texto claro.
- Hay un mensaje o página de confirmación contemplada para después del envío.
- El footer tiene tu nombre, el año y tus redes o correo.
- No hay ids duplicados ni ids con espacios o mayúsculas.
- Todas las etiquetas están cerradas y el código está indentado de forma legible.
- Abriste el archivo en el navegador, hiciste clic en cada enlace del menú y todos saltan a su sección.
- Intentaste enviar el formulario vacío y el navegador bloqueó el envío con sus avisos.
- Le pediste a alguien que lo vea y te diga si entiende quién eres y cómo contactarte en menos de un minuto.

Si marcaste todo, publícalo. Si algo falló, arréglalo hoy, no "después". El "después" es donde mueren los portafolios.

Graduación: ya dominas HTML5

Respira. Léelo otra vez: ya dominas HTML5. No significa que lo sepas todo (nadie lo sabe todo), significa que tienes las bases sólidas para construir páginas web reales, semánticas y funcionales. Entraste a este curso sin saber qué era una etiqueta y sales con un portafolio publicado en internet. Esa es la definición práctica de dominar un lenguaje: poder construir algo útil con él sin que nadie te lleve de la mano.

Celebra de la forma que quieras, pero celebra. La mayoría de la gente que dice "voy a aprender a programar" se queda en el intento. Tú terminaste. Esa diferencia, entre intentar y terminar, es la que separa a los profesionales de los soñadores. Guárdate esta sensación, porque la vas a necesitar: cada tecnología nueva que aprendas empezará igual, con esa incomodidad del primer día, y terminará igual, con un proyecto funcionando.

Y ahora, con la medalla puesta, mira hacia adelante. Porque HTML5 fue el primer nivel del videojuego, no el juego completo. Lo que sigue es igual de emocionante, y ya tienes la mitad del camino mental recorrido: sabes cómo se aprende una tecnología web (teoría corta, práctica larga, proyecto final). Ese método te servirá para todo lo que viene.

Lo que sigue: CSS3 para darle estilo

Tu portafolio funciona, pero se ve... simple. Texto negro sobre fondo blanco, enlaces azules, botones grises. Y está bien: así se ven todas las páginas antes del CSS. Pero nadie se enamora de un esqueleto. CSS3 es lo que le pone piel, ropa, peinado y maquillaje a tu HTML.

Con CSS3 vas a poder pintar tu portafolio con tus colores, elegir tipografías bonitas, separar las secciones con espacios elegantes, hacer que tu menú se vea como el de un sitio profesional y, lo más importante, hacer que todo se adapte al celular con diseño responsive. Esa es la magia que convierte "una página que funciona" en "una página que impresiona".

La buena noticia: CSS se engancha directo a lo que ya sabes. Esos ids únicos que pusiste en tus secciones serán los ganchos para pintar cada una. Esas classes que aprendiste en la unidad 8 serán tus uniformes de estilo. Todo el HTML semántico que escribiste con tanto cuidado hará que tu CSS sea más fácil, no más difícil. El que escribe buen HTML aprende CSS al doble de velocidad, porque ya piensa en estructura.

Tu siguiente curso en lucasegaacademy.com es el Curso Completo de CSS3, gratis como este. Treinta clases, misma metodología: teoría corta, práctica larga, proyecto final. Y tu proyecto será tomar este mismo portafolio y transformarlo. Imagina abrir tu `index.html` dentro de un mes y no reconocerlo de lo bonito que se ve, sabiendo que la estructura es la misma que construiste hoy. Esa es la promesa del CSS, y está a un curso de distancia.

Después: JavaScript y la interactividad

Después del estilo viene la vida. JavaScript es lo que hace que tu página reaccione: menús que se abren y se cierran, formularios que muestran la confirmación sin cambiar de página, galerías de fotos que se deslizan, validaciones con mensajes personalizados. Si HTML es el esqueleto y CSS la apariencia, JavaScript es el sistema nervioso: lo que siente y responde.

Con JavaScript tu formulario de contacto pasará de "se envía y ojalá funcione" a "muestra una animación de éxito y un mensaje personalizado con tu nombre". Tu navegación por anclas pasará de "salta de golpe" a "se desliza suavemente". Tu blog podrá cargar más artículos sin recargar la página. Pequeños detalles que el visitante no sabe nombrar pero sí sabe sentir: "esta página se siente profesional".

No te adelantes: JavaScript es más exigente porque es programación de verdad, con lógica, variables y decisiones. Pero llegarás con ventaja, porque ya entiendes el documento que JavaScript manipula. Cada id que pusiste, cada form que construiste, es un punto de agarre para tu futuro código. El camino natural es HTML5, luego CSS3, luego JavaScript, y en lucasegaacademy.com los tres cursos están diseñados para conectarse en ese orden.

Tu ruta frontend completa en lucasegaacademy.com

Ya eres frontend en formación. Frontend es todo lo que el usuario ve y toca en la web, y tú ya dominas su primera capa. La ruta completa se ve así, paso a paso, sin atajos mágicos.

Paso uno, completado: HTML5, la estructura. Ya lo tienes, con portafolio publicado.

Paso dos: CSS3, el estilo. Diseño, colores, responsive, animaciones básicas. Tu portafolio se vuelve visualmente profesional.

Paso tres: JavaScript, la interactividad. Tu página cobra vida y responde al usuario.

Paso cuatro: profundización. Accesibilidad avanzada, buenas prácticas, herramientas modernas, y tu primer framework cuando domines las bases. Pero eso es después; no corras antes de caminar.

Todos estos cursos están gratis en lucasegaacademy.com, con la misma metodología que ya te funcionó: clases cortas, práctica constante y proyecto final publicable. No necesitas pagar miles por un bootcamp para ser frontend; necesitas lo que ya demostraste tener: constancia para terminar lo que empiezas. Si terminaste este curso, puedes terminar los que siguen.

Cómo seguir practicando sin estancarte

El mayor riesgo después de graduarte no es olvidar etiquetas; es dejar de practicar. El conocimiento sin uso se oxida en semanas. Estas son tus reglas de mantenimiento, simples y realistas.

Primera: un proyecto pequeño por semana. No una página enorme; algo de una tarde. Una tarjeta de presentación digital, una receta de cocina con tabla de ingredientes, una lista de tus películas favoritas con imágenes. Lo importante es no soltar el teclado. Doce mini proyectos al año te enseñan más que doce tutoriales vistos sin practicar.

Segunda: clona páginas que te gusten. Elige un sitio sencillo y trata de replicar su estructura solo con HTML (sin copiar su código). Es el ejercicio favorito de los profesionales porque te enseña a "leer" páginas con ojos de constructor: ¿esto es un header? ¿esto una lista? ¿esto

un form? Cuando puedas desarmar mentalmente cualquier página, tu nivel sube sin que te des cuenta.

Tercera: escribe en tu blog cada semana. Un post corto sobre lo que aprendiste, lo que te costó y cómo lo resolviste. Escribir fija el conocimiento y, de paso, mantiene vivo tu portafolio. Los reclutadores adoran ver actividad reciente: dice "esta persona sigue aprendiendo".

Cuarta: relee tu propio código cada mes. Abre tu portafolio dentro de treinta días y seguro encontrarás algo que hoy harías mejor. Esa incomodidad es la medida exacta de tu progreso. Si tu código viejo te parece perfecto, preocúpate; si te parece mejorable, felicítate.

Quinta: enseña a alguien más. Explícale a un amigo qué es una etiqueta, ayúdale a hacer su primera página. Enseñar es el examen final que nunca termina: cada duda que resuelves te hace más sólido. Además, la comunidad que enseña es la comunidad que contrata.

Dónde publicar tu portafolio: GitHub Pages explicado sencillo

Tu portafolio en tu computadora solo lo ves tú. Para que el mundo lo vea necesitas publicarlo, y la forma más sencilla y gratuita es GitHub Pages. Te lo explico sin tecnicismos, paso a paso.

GitHub es una plataforma donde los programadores guardan su código, como una nube para proyectos. GitHub Pages es su función gratuita para publicar páginas web directamente desde ahí. No pagas hosting, no configuras servidores: subes tu index.html y te dan una dirección pública.

El proceso, en palabras simples, es este. Primero, crea una cuenta gratuita en github.com. Segundo, crea un repositorio nuevo (piensa en "repositorio" como "carpeta en la nube") con un nombre como "mi-portafolio". Tercero, sube tu archivo index.html a ese repositorio; GitHub tiene un botón para subir archivos, no necesitas comandos raros al inicio. Cuarto, entra a la configuración del repositorio, busca la sección "Pages" y activa la publicación desde la rama principal. Quinto, espera un par de minutos y GitHub te dará tu dirección pública, algo como tunombre.github.io/mi-portafolio.

Esa dirección es tu carta de presentación digital. Ponla en tu currículum, en tu perfil de redes profesionales, en tu firma de correo. Cuando alguien te pregunte "¿y tú qué sabes hacer?", no le cuentes: mándale tu liga. Eso es lo que hace un profesional.

Más adelante aprenderás a subir cambios con comandos y a conectar tu propio dominio, pero no lo necesitas hoy. Hoy necesitas estar publicado. Un portafolio publicado y sencillo vale infinitamente más que un portafolio perfecto guardado en tu disco duro.

Consejos para conseguir tus primeros proyectos

Tienes la habilidad y tienes la vitrina. Ahora, ¿cómo llegan los primeros clientes o el primer empleo? No hay magia, hay método. Estos consejos son para alguien que va empezando, sin contactos y sin experiencia laboral previa.

Primero, tu portafolio es tu mejor vendedor: trátalo como tal. Mantenlo actualizado, sin links rotos y con tus mejores proyectos al frente. Cuando apliques a algo, manda la liga directa a la sección relevante, no solo a la página principal. "Mira mis proyectos" con la liga a #proyectos dice más que mil palabras.

Segundo, empieza con tu círculo cercano. Negocios de familiares, amigos con emprendimientos, organizaciones de tu colonia: casi todos necesitan presencia web y casi ninguno puede pagar una agencia. Ofrece hacerles una página sencilla a precio accesible o incluso gratis las primeras veces, a cambio de testimonio y permiso para mostrarla en tu portafolio. Tus primeros tres clientes casi siempre salen de aquí.

Tercero, únete a comunidades. Grupos de desarrolladores en redes sociales, foros, eventos locales. No entres a pedir trabajo el primer día: entra a ayudar, a responder dudas de HTML, a mostrar tus mini proyectos. La gente contrata a quien ya conoce y en quien confía. Tu blog del portafolio te sirve aquí: compartir lo que aprendes te vuelve visible.

Cuarto, las plataformas de freelance son una opción, pero con estrategia. Al inicio compites con miles; tu ventaja es un portafolio real publicado y comunicación clara en español. Empieza con proyectos pequeños, entrégalos impecables y pide reseña siempre. Las primeras cinco reseñas valen más que cualquier certificado.

Quinto, nunca dejes de aprender en público. Sigue tomando los cursos de lucasegaacademy.com, publica tu progreso, celebra cada proyecto terminado. El mercado no contrata al que "algún día estará listo"; contrata al que demuestra, semana a semana, que avanza. Tú ya demostraste que terminas lo que empiezas. Esa es tu ventaja competitiva, úsala.

Ejemplos paso a paso

Ejemplo 1: sección "sobre mí" completa

Esta es la sección con la que todo visitante se forma su primera impresión de ti. Lleva tu foto, quién eres en dos o tres líneas honestas y tu lista de habilidades. Fíjate en tres cosas: el `id="sobre-mi"` sin espacios ni mayúsculas, el `h2` que titula la sección, y el `alt` de la imagen que describe la foto para quien no puede verla. La lista de habilidades usa `ul` porque el orden no importa: ninguna habilidad es "la número uno".

```
<!-- SECCIÓN SOBRE MÍ: tu presentación profesional -->
<section id="sobre-mi">
  <!-- Título de la sección: siempre un h2 -->
  <h2>Sobre mí</h2>

  <!-- Tu foto: el alt describe la imagen, nunca lo dejes vacío -->
  

  <!-- Quién eres: 2 o 3 líneas honestas, sin adornos -->
  <p>Hola, soy Ana. Estoy aprendiendo desarrollo web y me apasiona construir páginas claras y funcionales.</p>
  <p>Terminé el Curso Completo de HTML5 en Lucas EGA Academy y ahora estoy aprendiendo CSS3.</p>

  <!-- Tus habilidades: lista desordenada, una por li -->
  <h3>Mis habilidades</h3>
  <ul>
    <li>HTML5 semántico</li>
    <li>Formularios con validación</li>
    <li>Estructura de portafolios web</li>
    <li>Trabajo constante y ganas de aprender</li>
  </ul>
</section>
```

Si copias esta estructura y cambias los textos por los tuyos, ya tienes tu primera sección lista. La honestidad aquí vale oro: "estoy aprendiendo" bien dicho inspira más confianza que "experto" mal sostenido.

Ejemplo 2: artículo de blog con aside de links útiles

Aquí ves el artículo viviendo dentro de su sección "blog", con el aside anidado dentro del artículo porque esos links complementan este post específico. Observa la fecha de publicación: le da vida y contexto al contenido. El aside lleva su propio `h3` porque es una mini-sección dentro del artículo.

```

<!-- SECCIÓN BLOG: agrupa tus artículos -->
<section id="blog">
  <h2>Blog</h2>

  <!-- ARTICLE: contenido independiente, se entiende solo -->
  <article>
    <!-- Título del post y fecha de publicación -->
    <h3>Lo que aprendí con mi primer formulario</h3>
    <p>Publicado el 19 de septiembre de 2026</p>

    <!-- Contenido real: cuenta tu experiencia -->
    <p>Al principio pensé que un formulario era solo cajas de texto. Después descubrí los labels, la validación con required y el atributo name, y entendí que un buen formulario cuida al usuario en cada campo.</p>
    <p>Mi error favorito fue olvidar el name en los inputs: todo se veía bien, pero los datos nunca llegaban. Ahora lo reviso siempre dos veces.</p>

    <!-- ASIDE: complementa al article, no es lo principal -->
    <aside>
      <h4>Links útiles</h4>
      <ul>
        <li><a href="https://lucasegaacademy.com">Lucas EGA Academy</a></li>
        <li><a href="https://developer.mozilla.org/es/">MDN Web Docs en español</a></li>
      </ul>
    </aside>
  </article>
</section>

```

Cuando publiques tu segundo post, duplicas el article completo, cambias el contenido y listo. Esa es la belleza de las piezas independientes: crecen sin romper nada.

Ejemplo 3: formulario de contacto final completo

Este es el formulario de la clase 29, con todo aplicado: labels conectados, validación con required y type="email", textarea con tamaño y mínimo de caracteres, y botón de envío con texto humano. Debajo va el mensaje de confirmación que promete respuesta. Léelo de arriba a abajo y verifica cada pareja label-for con su id.

```

<!-- SECCIÓN CONTACTO: tu puerta de entrada profesional -->
<section id="contacto">
  <h2>Contacto</h2>
  <p>¿Tienes un proyecto en mente? Escíbeme y te respondo en menos de 48 horas.</p>

  <!-- FORM: action indica a dónde van los datos, method post los protege -->
  <form action="/enviar" method="post">

    <!-- Campo nombre: texto simple, obligatorio, mínimo 2 letras -->
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required minlength="2"
placeholder="Tu nombre">

    <!-- Campo correo: type email valida el formato, required lo exige -->
    <label for="email">Correo electrónico:</label>
    <input type="email" id="email" name="email" required
placeholder="tucorreo@ejemplo.com">

    <!-- Campo mensaje: textarea amplio, mínimo 10 caracteres -->
    <label for="mensaje">Mensaje:</label>
    <textarea id="mensaje" name="mensaje" rows="5" cols="40" required
minlength="10" placeholder="Cuéntame en qué puedo ayudarte..."></textarea>

    <!-- Botón de envío: type submit lo activa -->
    <button type="submit">Enviar mensaje</button>
  </form>

  <!-- Confirmación: el visitante sabe qué pasará después -->
  <p>Al enviar recibirás una confirmación. ¡Gracias por tu mensaje!</p>
</section>

```

Prueba este formulario en tu navegador: envíalo vacío y mira cómo el navegador bloquea el envío; escribe un correo sin arroba y mira el aviso. Esa validación gratis es tu guardia de seguridad trabajando.

Ejemplo 4: portafolio completo integrando todo

Este es el ejemplo mayor: todas las piezas ensambladas en un solo archivo index.html. Léelo como un mapa: head con sus metas, header con nav y anclas, main con las cuatro secciones, y footer cerrando. Cada comentario marca una zona. Cuando armes el tuyo, usa este como esqueleto y cambia los contenidos.

```

<!DOCTYPE html>
<!-- Documento HTML5 en español -->
<html lang="es">
<head>
  <!-- Metas básicas: caracteres, celular y descripción -->
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta name="description" content="Portafolio de Ana, desarrolladora web en formación.">
  <title>Ana | Desarrolladora web en formación</title>
</head>
<body>

  <!-- HEADER: presentación y navegación -->
  <header>
    <h1>Ana García</h1>
    <p>Desarrolladora web en formación</p>
    <!-- NAV con anclas: cada enlace salta a su sección -->
    <nav>
      <ul>
        <li><a href="#sobre-mi">Sobre mí</a></li>
        <li><a href="#proyectos">Proyectos</a></li>
        <li><a href="#blog">Blog</a></li>
        <li><a href="#contacto">Contacto</a></li>
      </ul>
    </nav>
  </header>

  <!-- MAIN: el contenido principal de la página -->
  <main>

    <!-- SECCIÓN SOBRE MÍ -->
    <section id="sobre-mi">
      <h2>Sobre mí</h2>
      
      <p>Hola, soy Ana. Construyo páginas web claras y funcionales con HTML5.</p>
      <h3>Mis habilidades</h3>
      <ul>
        <li>HTML5 semántico</li>
        <li>Formularios con validación</li>
        <li>Publicación con GitHub Pages</li>
      </ul>
    </section>
  </main>

```



```

<!-- SECCIÓN PROYECTOS -->
<section id="proyectos">
  <h2>Proyectos</h2>
  <article>
    <h3>Mi portafolio personal</h3>
    <p>Página con secciones semánticas, blog y formulario de contacto
validado.</p>
  </article>
  <article>
    <h3>Página de recetas</h3>
    <p>Recetario con tablas de ingredientes, listas de pasos e imágenes.</p>
  </article>
</section>

<!-- SECCIÓN BLOG con article y aside -->
<section id="blog">
  <h2>Blog</h2>
  <article>
    <h3>Lo que aprendí con mi primer formulario</h3>
    <p>Publicado el 19 de septiembre de 2026</p>
    <p>Descubrí que un buen formulario cuida al usuario en cada campo: labels
claros, validación amable y confirmación al enviar.</p>
    <aside>
      <h4>Links útiles</h4>
      <ul>
        <li><a href="https://lucasegaacademy.com">Lucas EGA Academy</a></li>
      </ul>
    </aside>
  </article>
</section>

<!-- SECCIÓN CONTACTO con formulario final -->
<section id="contacto">
  <h2>Contacto</h2>
  <form action="/enviar" method="post">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required minlength="2">
    <label for="email">Correo electrónico:</label>
    <input type="email" id="email" name="email" required>
    <label for="mensaje">Mensaje:</label>
    <textarea id="mensaje" name="mensaje" rows="5" cols="40" required
minlength="10"></textarea>
    <button type="submit">Enviar mensaje</button>
  </form>

```

```
        <p>Al enviar recibirás una confirmación. ¡Gracias por tu mensaje!</p>
    </section>

</main>

<!-- FOOTER: cierre con tu firma -->
<footer>
    <p>Ana García, 2026. Hecho con HTML5 aprendido en Lucas EGA Academy.</p>
</footer>

</body>
</html>
```

Guarda esto como index.html, ábrelo en tu navegador y navega con el menú. Si cada enlace salta a su sección y el formulario válida, tienes un portafolio funcionando. Lo demás es personalizarlo hasta que suene a ti.

Práctica

Ejercicio 1: tu sección "sobre mí"

Enunciado: crea un archivo sobre-mi.html que contenga únicamente la sección "sobre mí" de tu portafolio. Debe llevar id="sobre-mi", un h2 con el título, tu foto con alt descriptivo, dos párrafos donde te presentes con honestidad y una lista desordenada con al menos cuatro habilidades reales que tengas hoy.

Solución paso a paso. Paso 1: abre tu editor y crea el archivo sobre-mi.html. Paso 2: escribe la etiqueta section con el id en minúsculas y con guion: section id="sobre-mi". Recuerda: sin espacios, sin mayúsculas, sin tildes en el id. Paso 3: dentro, escribe el h2 con el texto "Sobre mí" (aquí sí van tildes, porque es texto visible). Paso 4: agrega tu imagen con img, src apuntando a tu foto y alt describiendo quién aparece en ella. Si aún no tienes foto, usa una imagen temporal pero escribe el alt como si ya fuera la definitiva. Paso 5: escribe dos párrafos honestos: quién eres, qué estás aprendiendo y qué te gusta construir. Nada de "experto apasionado por la innovación": escribe como hablas. Paso 6: agrega un h3 "Mis habilidades" y una ul con cuatro li, uno por habilidad. Solo habilidades que puedas demostrar: "HTML5 semántico" sí, "inteligencia artificial avanzada" no. Paso 7: guarda, abre en el navegador y verifica que todo se vea y que el código esté indentado.

```

<section id="sobre-mi">
  <h2>Sobre mí</h2>
  
  <p>Hola, soy Luis. Terminé el curso de HTML5 y estoy construyendo mi portafolio
para conseguir mis primeros proyectos.</p>
  <p>Me gusta el código ordenado y las páginas que cargan rápido y se entienden
solas.</p>
  <h3>Mis habilidades</h3>
  <ul>
    <li>HTML5 semántico</li>
    <li>Formularios con validación</li>
    <li>Estructura de portafolios</li>
    <li>Publicación con GitHub Pages</li>
  </ul>
</section>

```

Ejercicio 2: navegación por anclas que sí funciona

Enunciado: crea un archivo menú.html con un nav que contenga cuatro enlaces: "Sobre mí", "Proyectos", "Blog" y "Contacto". Debajo del nav, crea las cuatro secciones correspondientes, cada una con su id y un h2. Luego abre el archivo y comprueba que cada enlace salta a su sección.

Solución paso a paso. Paso 1: crea menú.html con la estructura básica (DOCTYPE, html lang="es", head, body). Paso 2: dentro del body escribe el nav con una ul de cuatro li; cada a lleva href con gato más el id en minúsculas: #sobre-mi, #proyectos, #blog, #contacto. Paso 3: debajo del nav escribe las cuatro sections, cada una con su id idéntico al del enlace (letra por letra) y un h2 con el título visible. Paso 4: para que el salto se note, escribe dos o tres párrafos de relleno en cada sección; si las secciones están pegadas, el navegador "salta" pero no lo percibes. Paso 5: guarda y abre en el navegador. Haz clic en cada enlace del menú y observa el desplazamiento. Paso 6: si algún enlace no salta, compara el href con el id: el error más común es una letra de diferencia (#proyecto sin s contra id="proyectos"). Corrige y vuelve a probar hasta que los cuatro funcionen.

```

<nav>
  <ul>
    <li><a href="#sobre-mi">Sobre mí</a></li>
    <li><a href="#proyectos">Proyectos</a></li>
    <li><a href="#blog">Blog</a></li>
    <li><a href="#contacto">Contacto</a></li>
  </ul>
</nav>

<section id="sobre-mi">
  <h2>Sobre mí</h2>
  <p>Contenido de presentación...</p>
  <p>Más contenido para dar altura a la sección...</p>
</section>

<section id="proyectos">
  <h2>Proyectos</h2>
  <p>Contenido de proyectos...</p>
  <p>Más contenido para dar altura a la sección...</p>
</section>

<section id="blog">
  <h2>Blog</h2>
  <p>Contenido del blog...</p>
  <p>Más contenido para dar altura a la sección...</p>
</section>

<section id="contacto">
  <h2>Contacto</h2>
  <p>Contenido de contacto...</p>
  <p>Más contenido para dar altura a la sección...</p>
</section>

```

Ejercicio 3: tu primer artículo con aside

Enunciado: dentro de una sección con id="blog", publica tu primer artículo real: un post donde cuentes qué fue lo más difícil de aprender HTML5 y cómo lo superaste. El artículo debe llevar título (h3), fecha de publicación, al menos dos párrafos de contenido honesto y un aside con el título "Links útiles" y dos enlaces que te hayan servido durante el curso.

Solución paso a paso. Paso 1: crea blog.html o trabaja sobre tu archivo del ejercicio 2. Paso 2: escribe section id="blog" con su h2 "Blog". Paso 3: dentro, abre el artículo y escribe el h3 con un

título atractivo, no genérico: "Cómo dejé de pelearme con los formularios" vende más que "Mi primer post". Paso 4: agrega la fecha en un párrafo corto. Paso 5: escribe dos párrafos de verdad: qué te costó (¿las tablas? ¿los labels? ¿los ids?), qué hiciste para entenderlo y qué consejo le darías a quien va empezando. Si te sale de corazón, está bien escrito. Paso 6: dentro del article, al final, agrega el aside con un h4 "Links útiles" y una ul con dos li, cada uno con un a a un recurso real que hayas usado (por ejemplo, lucasegaacademy.com y MDN). Paso 7: verifica que el aside esté dentro del article (porque complementa a ese post) y no flotando fuera. Paso 8: léelo en voz alta; si suena a ti, publícalo en tu portafolio.

```
<section id="blog">
  <h2>Blog</h2>
  <article>
    <h3>Cómo dejé de pelearme con los formularios</h3>
    <p>Publicado el 19 de septiembre de 2026</p>
    <p>Lo más difícil para mí fueron los formularios. Entendía cada etiqueta por separado, pero al juntarlas algo siempre fallaba: el label no activaba el campo, el botón enviaba cuando no debía, los datos no llegaban.</p>
    <p>Lo superé armando el mismo formulario cinco veces desde cero, sin copiar. A la quinta ya no pensaba en las etiquetas: pensaba en el usuario que lo iba a llenar. Mi consejo: si algo no te sale, repítelo hasta que te salga dormido.</p>
    <aside>
      <h4>Links útiles</h4>
      <ul>
        <li><a href="https://lucasegaacademy.com">Lucas EGA Academy</a></li>
        <li><a href="https://developer.mozilla.org/es/">MDN Web Docs en español</a></li>
      </ul>
    </aside>
  </article>
</section>
```

Ejercicio 4: formulario de contacto con validación y confirmación

Enunciado: crea contacto.html con la sección de contacto completa: h2, un párrafo de invitación, un form con action="/enviar" y method="post" que pida nombre (requerido, mínimo 2 caracteres), correo (requerido, tipo email) y mensaje (requerido, mínimo 10 caracteres), cada campo con su label conectado, y un botón de envío. Debajo, un párrafo de confirmación. Luego prueba la validación: intenta enviarlo vacío, con un correo inválido y con un mensaje de una sola palabra.

Solución paso a paso. Paso 1: crea contacto.html con estructura básica. Paso 2: escribe section id="contacto" con h2 y el párrafo de invitación. Paso 3: abre el form con action="/enviar" y method="post". Paso 4: agrega el campo nombre: label for="nombre", input type="text" id="nombre" name="nombre" required minlength="2". Verifica que el for y el id sean idénticos y que exista el name. Paso 5: agrega el campo correo igual, pero con type="email". Paso 6: agrega el mensaje con textarea: id="mensaje" name="mensaje" rows="5" cols="40" required minlength="10", con su label. Recuerda que textarea lleva etiqueta de cierre. Paso 7: agrega button type="submit" con texto "Enviar mensaje". Paso 8: cierra el form y escribe el párrafo de confirmación. Paso 9: abre en el navegador y haz las tres pruebas: envío vacío (el navegador debe bloquearlo), correo sin arroba (debe avisar del formato) y mensaje de una palabra (debe pedir más caracteres). Si las tres pruebas bloquean correctamente, tu validación funciona. Paso 10: revisa una vez más que ningún input se quedó sin name: es el error invisible más común.

```
<section id="contacto">
  <h2>Contacto</h2>
  <p>¿Tienes un proyecto en mente? Escríbeme.</p>
  <form action="/enviar" method="post">
    <label for="nombre">Nombre:</label>
    <input type="text" id="nombre" name="nombre" required minlength="2"
placeholder="Tu nombre">
    <label for="email">Correo electrónico:</label>
    <input type="email" id="email" name="email" required
placeholder="tucorreo@ejemplo.com">
    <label for="mensaje">Mensaje:</label>
    <textarea id="mensaje" name="mensaje" rows="5" cols="40" required
minlength="10" placeholder="Cuéntame tu idea..."></textarea>
    <button type="submit">Enviar mensaje</button>
  </form>
  <p>Al enviar recibirás una confirmación. ¡Gracias por escribirme!</p>
</section>
```

Ejercicio 5: proyecto final, arma tu portafolio completo

Enunciado: integra todo lo anterior en un solo archivo index.html: documento completo con DOCTYPE, head con metas y title, header con tu h1 y nav de cuatro anclas, main con las secciones sobre-mí, proyectos (con dos articles), blog (con un article y su aside) y contacto (con el formulario validado), y footer con tu firma. Luego pásalo por el checklist final de esta unidad y publícalo con GitHub Pages.

Solución paso a paso. Paso 1: crea una carpeta "mi-portafolio" y dentro el archivo index.html. También guarda ahí tu foto con un nombre simple como "mi-foto.jpg". Paso 2: escribe la base: DOCTYPE, html lang="es", head con meta charset="UTF-8", meta viewport, meta description y title con tu nombre. Paso 3: en el body, escribe el header: tu h1 con tu nombre, un párrafo con tu título profesional y el nav con los cuatro enlaces de ancla (verifica cada href contra su id). Paso 4: abre el main y pega tu sección sobre-mí del ejercicio 1, adaptada y mejorada. Paso 5: agrega la sección proyectos con id="proyectos": un h2 y dos articles, cada uno con h3 y un párrafo describiendo un proyecto real que hayas hecho (aunque sea un ejercicio del curso bien presentado). Paso 6: agrega tu sección blog del ejercicio 3 con tu post real y su aside. Paso 7: agrega tu formulario de contacto del ejercicio 4, completo y validado. Paso 8: cierra el main y escribe el footer con tu nombre, el año y una línea sobre dónde aprendiste. Paso 9: abre el archivo en el navegador y recorre el checklist final punto por punto: un solo h1, ids únicos, anclas que saltan, formulario que válida, etiquetas cerradas. Corrige todo lo que falle. Paso 10: crea tu cuenta de GitHub, sube la carpeta como repositorio, activa GitHub Pages y consigue tu liga pública. Paso 11: mándale tu liga a una persona de confianza y pídele que te diga en una frase quién eres y cómo contactarte después de verla 60 segundos. Si lo logra, tu portafolio comunica. Si no, ajusta los textos y repite.

Este ejercicio no tiene una única solución correcta: la tienes tú, con tus palabras y tus proyectos. Usa el ejemplo 4 de esta unidad como esqueleto, pero el contenido debe sonar a ti. Cuando termines, habrás hecho lo que pocos logran: empezar y terminar. Actualiza tu portafolio cada mes, sigue con CSS3 en lucasegaacademy.com y no sueltes el ritmo. Nos vemos en el siguiente curso.

Resumen de la unidad

- Tu proyecto final es un portafolio publicable: tu aparador profesional, no un ejercicio más.
- La etiqueta section agrupa contenido por tema; cada sección importante lleva un id único, en minúsculas y con guiones.
- El article envuelve contenido independiente que se entiende solo: cada post de tu blog es un article.
- El aside complementa sin ser lo principal: links útiles dentro del article o barra lateral de la página.
- La navegación por anclas conecta tu menú con tus secciones: href="#id" salta al elemento con ese id, sin recargar.

- El formulario de contacto usa form con action y method="post": tres campos esenciales (nombre, correo, mensaje), cada uno con label, id y name.
- La validación de HTML5 trabaja gratis para ti: required, type="email", minlength y maxlength, con avisos automáticos del navegador.
- Todo formulario necesita un mensaje de confirmación: el visitante nunca debe quedarse con la duda de si su mensaje llegó.
- Recorriste las 30 clases: estructura, texto, listas, enlaces, imágenes, tablas, formularios, semántica, multimedia, buenas prácticas, SEO, anclas y validación.
- El checklist final es tu control de calidad: revísalo punto por punto antes de publicar.
- Ya dominas HTML5: puedes construir páginas reales, semánticas y funcionales sin ayuda.
- Lo que sigue es CSS3 para el estilo y JavaScript para la interactividad, gratis en lucasegaacademy.com.
- Publica tu portafolio con GitHub Pages y pon tu liga en tu currículum, tus redes y tu firma de correo.
- Practica sin pausa: un mini proyecto por semana, clona páginas, escribe en tu blog y enseña a otros.
- Tus primeros proyectos salen de tu círculo cercano, las comunidades y el freelance con estrategia.
- La diferencia entre soñar y ser profesional es terminar lo que empiezas: tú ya lo demostraste.

EXAMEN FINAL

Curso Completo de HTML5 - Lucas EGA Academy (lucasegaacademy.com)

Instrucciones

Bienvenido a tu examen final. Si llegaste hasta aquí es porque ya viste las 30 clases del curso, así que este examen es tu oportunidad de demostrar lo que aprendiste y de detectar qué temas necesitas repasar. Tómallo en serio, pero sin miedo: nadie nació sabiendo HTML y cada error que encuentres aquí es una lección que ya no se te va a olvidar.

Cómo presentar el examen:

- Busca un lugar tranquilo donde puedas concentrarte sin interrupciones.
- Ten a la mano hojas blancas y pluma, o abre un editor de texto si prefieres escribir el código en tu computadora.
- No uses tus apuntes, el libro ni internet durante el examen. La idea es medir lo que realmente se te quedó en la cabeza.
- Lee cada pregunta completa antes de responder. Muchas se pierden por contestar a la carrera.
- En las preguntas de código, escribe el código completo aunque creas que es obvio. Un examen de HTML se responde con HTML, no con descripciones.
- En las preguntas abiertas de explicación, usa tus propias palabras. No se trata de memorizar definiciones, se trata de entender.

Reglas del examen:

- Tiempo sugerido: 90 minutos. Si terminas antes, usa el tiempo extra para revisar tus respuestas.
- El examen tiene 50 preguntas divididas en tres tipos: 20 de opción múltiple, 15 de verdadero o falso y 15 abiertas.
- En las preguntas de verdadero o falso debes escribir V o F y justificar tu respuesta en una sola línea. Sin justificación, la respuesta no cuenta aunque la letra sea correcta.
- En las preguntas abiertas de código debes encerrar tu respuesta en un bloque de código. En las de explicación, cada respuesta debe indicar los puntos que se evalúan.

- No te quedes atorado en una pregunta difícil. Márcala, sigue con las demás y regresa al final si te sobra tiempo.

Sistema de puntaje:

- Cada pregunta vale 2 puntos. No hay preguntas que valgan más que otras.
- 50 preguntas por 2 puntos dan un total de 100 puntos.
- La calificación mínima aprobatoria es 60 puntos.
- Las preguntas de opción múltiple y de verdadero o falso se califican como correctas o incorrectas, sin puntos parciales.
- Las preguntas abiertas se califican con criterio: 2 puntos si la respuesta está completa y correcta, 1 punto si está parcialmente correcta o le faltan elementos importantes, 0 puntos si está mal o vacía.

Tabla de calificaciones:

- 90 a 100 puntos: Excelente. Dominas HTML5 de verdad. Estás listo para el siguiente paso: CSS3.
- 80 a 89 puntos: Muy bien. Tienes una base sólida, repasa los temas donde fallaste y sigue adelante.
- 70 a 79 puntos: Bien. Aprobaste con margen, pero hay temas que todavía no dominas. Repásalos antes de pasar a CSS3.
- 60 a 69 puntos: Suficiente. Pasaste de panzazo. Te recomiendo repasar el curso completo una vez más antes de continuar.
- 0 a 59 puntos: No aprobado. No te desanimes: vuelve a las clases, practica con el proyecto del portafolio y presenta el examen de nuevo.

Rúbrica detallada para calificar las preguntas abiertas:

- Preguntas de código (36, 38, 39, 40, 41, 43, 45, 46, 47, 49 y 50): 2 puntos si el código usa las etiquetas y atributos correctos, con apertura, cierre y anidación adecuados; 1 punto si la idea general es correcta pero faltan atributos, hay errores de cierre o de anidación; 0 puntos si el código no resuelve lo pedido o está vacío. No se restan puntos por el orden de los atributos ni por los espacios en blanco.
- Preguntas de explicación (37, 42, 44 y 48): 2 puntos si la respuesta incluye todos los puntos clave indicados en la solución, explicados con tus propias palabras; 1 punto si incluye la idea central pero le faltan puntos clave; 0 puntos si la explicación es incorrecta o

no responde lo preguntado. No se evalúa la redacción perfecta, se evalúa que entiendas el concepto.

Mapa del examen: cada pregunta y la clase del curso que evalúa:

- Preguntas 1 y 37: Clase 1 (Qué es HTML5) y Clase 30 (tu siguiente paso)
- Preguntas 2, 21 y 36: Clase 2 (Estructura básica)
- Preguntas 3 y 38: Clase 3 (Etiquetas de texto)
- Preguntas 22 y 38: Clase 4 (Párrafos y saltos)
- Preguntas 4 y 39: Clase 5 (Enlaces)
- Preguntas 5, 23 y 40: Clase 6 (Imágenes)
- Preguntas 6 y 24: Clase 7 (Listas)
- Preguntas 7 y 41: Clase 8 (Tablas básicas)
- Preguntas 8 y 42: Clase 9 (Div y span)
- Preguntas 9 y 26: Clase 10 (Atributos globales)
- Preguntas 10 y 43: Clase 11 (Formularios)
- Preguntas 11, 27 y 43: Clase 12 (Inputs HTML5)
- Preguntas 12, 28 y 43: Clase 13 (Validación nativa)
- Preguntas 13, 29 y 44: Clase 14 (HTML semántico)
- Preguntas 14, 30 y 45: Clase 15 (Meta tags y SEO)
- Preguntas 15 y 46: Clase 16 (Audio y video)
- Preguntas 25 y 47: Clase 17 (Tablas avanzadas)
- Pregunta 31: Clase 18 (Iframe y contenido incrustado)
- Preguntas 16 y 32: Clase 19 (Canvas)
- Pregunta 17: Clase 20 (SVG)
- Pregunta 33: Clase 21 (Listas de definición)
- Pregunta 18: Clase 22 (Entidades y símbolos)
- Preguntas 19, 23 y 48: Clase 23 (Accesibilidad web)
- Preguntas 29, 34 y 44: Clase 24 (Buenas prácticas)
- Pregunta 49: Clase 25 (Formularios pro: fieldset y datalist)

- Preguntas 20 y 35: Clase 26 (Etiquetas obsoletas)
- Pregunta 50: Clases 27, 28 y 29 (Proyecto: tu portafolio)

Usa este mapa para repasar: si fallaste varias preguntas de la misma clase, ya sabes exactamente qué parte del curso volver a estudiar.

Consejos para aprovechar tu resultado:

- No te califiques con coraje ni con lástima: califícate con honestidad. El examen es una herramienta de diagnóstico, no un juicio sobre tu capacidad.
- Marca las preguntas que fallaste y escribe a un lado el número de clase correspondiente. En una semana vuelve a presentar solo esas preguntas y mide tu avance.
- Si aprobaste con más de 90 puntos, tu siguiente paso natural es el curso de CSS3 en lucasegaacademy.com: ahí le darás estilo, color y diseño a todo lo que hoy sabes estructurar.
- Si no aprobaste, no es el fin del mundo: es información valiosa. Repasa las clases marcadas, reconstruye el proyecto del portafolio con calma y vuelve a intentarlo. La constancia le gana al talento, siempre.

Recomendación final: cuando termines, califica tu examen con la sección de respuestas y soluciones. No te limites a contar aciertos: lee cada explicación, sobre todo la de las preguntas que fallaste. Ahí está el verdadero aprendizaje. El que repasa sus errores avanza el doble de rápido.

¡Mucho éxito! Demuestra de qué estás hecho.

Preguntas

PREGUNTA 1:

¿Qué es HTML5? A) Un lenguaje de programación para crear aplicaciones de escritorio B) La última versión del lenguaje de marcado con el que se construyen todas las páginas web C) Un programa para diseñar imágenes y logotipos D) Un navegador de internet moderno

PREGUNTA 2:

¿Cuál es la primera línea que debe llevar todo documento HTML5? A) <html> B) <head> C) <!DOCTYPE html> D) <body>

PREGUNTA 3:

¿Qué par de etiquetas se usa para un título principal grande y para darle importancia fuerte a un texto (negrita con significado)? A) <p> y B) <h1> y C) <title> y D) <header> y <mark>

PREGUNTA 4:

Quieres enlazar a Google (sitio externo) y a tu página contacto.html (sitio interno). ¿Cuál opción usa correctamente la etiqueta de enlaces? A) Google y Contacto B) <link src="https://google.com">Google</link> y <link src="contacto.html">Contacto</link> C) Google y Contacto D) <href="https://google.com">Google</href> y <href="contacto.html">Contacto</href>

PREGUNTA 5:

¿Qué atributo de la etiqueta es indispensable para la accesibilidad porque describe la imagen cuando esta no se puede ver? A) src B) width C) alt D) href

PREGUNTA 6:

¿Cuál es la diferencia entre y ? A) ul es para enlaces y ol es para imágenes B) ul crea una lista con viñetas (no ordenada) y ol crea una lista numerada (ordenada) C) ul es más moderna que ol y la reemplaza D) No hay diferencia, son sinónimos

PREGUNTA 7:

En una tabla, ¿qué etiqueta se usa para las celdas de encabezado? A) <td> B) <tr> C) <th> D) <thead>

PREGUNTA 8:

¿Cuál es la diferencia entre <div> y ? A) div es un contenedor de bloque y span es un contenedor en línea para fragmentos de texto B) div es para texto y span es para imágenes C) div está obsoleto y span es su reemplazo moderno D) Son exactamente lo mismo y se pueden intercambiar siempre

PREGUNTA 9:

¿Cuál es la diferencia entre los atributos globales id y class? A) id se puede repetir en varios elementos y class no B) id es único en toda la página y class se puede usar en varios elementos C) id es para CSS y class es para JavaScript D) No hay diferencia entre ellos

PREGUNTA 10:

Dentro de un formulario, ¿qué etiqueta se usa para crear el botón que envía los datos? A) `<input type="text">` B) `<send>` C) `<button>` D) `<submit>`

PREGUNTA 11:

¿Qué tipo de input válida automáticamente que el texto tenga formato de correo electrónico? A) `<input type="text">` B) `<input type="email">` C) `<input type="mail">` D) `<input type="correo">`

PREGUNTA 12:

¿Qué atributo obliga al usuario a llenar un campo antes de enviar el formulario, usando solo el navegador y sin JavaScript? A) placeholder B) required C) value D) name

PREGUNTA 13:

¿Cuál de estas etiquetas NO es semántica? A) `<article>` B) `<nav>` C) `<div>` D) `<footer>`

PREGUNTA 14:

¿Para qué sirve la meta etiqueta viewport? A) Para cambiar el color de fondo de la página B) Para que la página se vea bien en celulares ajustando el ancho al dispositivo C) Para poner el título que aparece en Google D) Para acelerar la carga de las imágenes

PREGUNTA 15:

¿Qué atributo necesitas en las etiquetas `<audio>` y `<video>` para que el usuario pueda reproducir, pausar y ajustar el volumen? A) autoplay B) controls C) play D) media

PREGUNTA 16:

¿Qué se necesita para dibujar dentro de un `<canvas>`? A) Nada más, la etiqueta `<canvas>` dibuja gráficos por sí sola B) JavaScript, porque canvas es un lienzo vacío que se dibuja con código C) La etiqueta `<draw>` dentro del canvas D) Un archivo de imagen como fondo

PREGUNTA 17:

¿Cuál es la principal ventaja de SVG frente a Canvas? A) SVG es más rápido para videojuegos con muchos gráficos B) Los gráficos SVG son vectoriales y no se pixelan al agrandarlos C) Canvas no necesita JavaScript y SVG sí D) SVG solo sirve para fotografías

PREGUNTA 18:

¿Qué entidad HTML muestra el símbolo `<` como texto sin que el navegador lo interprete como una etiqueta? A) ` ` B) `<` C) `&` D) `©`

PREGUNTA 19:

¿Para qué sirve el atributo alt en una imagen, además de mostrar un texto si la imagen no carga? A) Para hacer que la imagen se vea más grande B) Para que los lectores de pantalla describan la imagen a personas con discapacidad visual C) Para cambiar el formato del archivo de imagen D) No tiene ningún otro uso

PREGUNTA 20:

¿Cuál de estas etiquetas está obsoleta en HTML5 y NO debes usarla? A) <section> B) <marquee> C) <video> D) <article>

PREGUNTA 21:

La declaración <!DOCTYPE html> es opcional y la página funciona exactamente igual sin ella. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 22:

La etiqueta
 necesita su etiqueta de cierre </br> para que el salto de línea funcione. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 23:

Puedes publicar una imagen sin el atributo alt y no pasa nada grave. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 24:

La etiqueta puede usarse sola, fuera de o , y el código sigue siendo válido. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 25:

La etiqueta <th> solo puede usarse dentro de <thead> y en ningún otro lugar de la tabla. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 26:

El atributo class debe ser único en toda la página, igual que el atributo id. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 27:

El input `<input type="password">` oculta lo que el usuario escribe mostrando puntos o asteriscos. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 28:

El atributo `required` válida el campo usando solo el navegador, sin necesidad de escribir JavaScript. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 29:

Las etiquetas `<div>` y `<section>` son exactamente lo mismo y se pueden intercambiar en cualquier caso. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 30:

La meta etiqueta `viewport` solo sirve para computadoras de escritorio y no tiene efecto en celulares. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 31:

Con la etiqueta `<iframe>` puedes incrustar un video de YouTube dentro de tu propia página. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 32:

La etiqueta `<canvas>` dibuja gráficos por sí sola, sin necesidad de JavaScript. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 33:

En una lista de definición, las etiquetas `<dt>` y `<dd>` siempre van dentro de `<dl>`. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 34:

Comentar tu código con `<!-- -->` es una buena práctica recomendada para que otros entiendan tu trabajo. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 35:

La etiqueta `<marquee>` sigue siendo recomendada en HTML5 para crear textos en movimiento. Escribe V o F y justifica tu respuesta en una línea.

PREGUNTA 36:

Escribe en un bloque de código la estructura básica completa de un documento HTML5: la declaración DOCTYPE, la etiqueta html, el head con el meta charset UTF-8 y un title, y el body con un encabezado h1 que diga "Hola Mundo". Se evalúa: el orden correcto de las etiquetas, la anidación adecuada y que cada etiqueta de apertura tenga su cierre.

PREGUNTA 37:

Explica con tus palabras qué es HTML5, para qué sirve y cuál es el siguiente paso recomendado después de dominarlo. Se evalúa: que menciones que es un lenguaje de marcado (no un lenguaje de programación), que sirve para construir la estructura de las páginas web, y que el siguiente paso es aprender CSS3 para darle estilo.

PREGUNTA 38:

Escribe en un bloque de código un párrafo que contenga: texto normal, una palabra en negrita con , una palabra en cursiva con , un salto de línea con
 y, fuera del párrafo, una línea separadora con <hr>. Se evalúa: el uso correcto de cada etiqueta en su lugar correspondiente.

PREGUNTA 39:

Escribe en un bloque de código dos enlaces: uno externo a <https://lucasegaacademy.com> que abra en una pestaña nueva del navegador, y uno interno a [contacto.html](#) que abra en la misma pestaña. Se evalúa: la etiqueta <a> con el atributo href correcto en ambos casos y el atributo target="_blank" solo en el enlace externo.

PREGUNTA 40:

Escribe en un bloque de código una imagen con src="foto.jpg", un texto alternativo descriptivo, ancho de 300 y alto de 200. Se evalúa: la etiqueta con los cuatro atributos correctos: src, alt, width y height.

PREGUNTA 41:

Escribe en un bloque de código una tabla con una fila de encabezado de dos columnas (Nombre y Edad) y dos filas de datos con la información de dos personas. Se evalúa: las etiquetas table, tr, th para los encabezados y td para los datos, todo correctamente abierto y cerrado.

PREGUNTA 42:

Explica con tus palabras la diferencia entre <div> y y da un ejemplo práctico de cuándo usarías cada uno. Se evalúa: que expliques que div es un contenedor de bloque y

span es un contenedor en línea, con ejemplos de uso correctos para cada uno. Un tip: cuando dudes entre div y span, mira el contexto. ¿Agrupas bloques que van uno debajo de otro? Usa div. ¿Marcas palabras dentro de una línea de texto? Usa span. Y antes de usar div, pregúntate si hay una etiqueta semántica que describa mejor ese bloque.

PREGUNTA 43:

Escribe en un bloque de código un formulario que incluya: un campo de texto para el nombre con texto de ayuda (placeholder), un campo de email obligatorio y un botón para enviar el formulario. Se evalúa: la etiqueta form, los inputs con type="text" y type="email", el atributo required en el email, el placeholder y el botón de envío.

PREGUNTA 44:

Explica con tus palabras qué es el HTML semántico, menciona al menos cuatro etiquetas semánticas y explica por qué a Google le gusta este tipo de código. Se evalúa: el concepto de que las etiquetas describen el significado del contenido, cuatro etiquetas semánticas correctas y la relación con el posicionamiento en buscadores (SEO).

PREGUNTA 45:

Escribe en un bloque de código los tres meta tags esenciales que debe llevar toda página HTML5 moderna en su head. Se evalúa: el meta charset UTF-8, el meta name="description" y el meta name="viewport", cada uno con su sintaxis correcta.

PREGUNTA 46:

Escribe en un bloque de código un reproductor de video con controles, que cargue el archivo "video.mp4" e incluya un mensaje de respaldo para navegadores que no soporten video. Se evalúa: la etiqueta video con el atributo controls, la fuente del archivo y el texto alternativo de respaldo dentro de la etiqueta.

PREGUNTA 47:

Escribe en un bloque de código una tabla que use thead y tbody, donde la fila de encabezado tenga una celda que abarque dos columnas usando colspan. Se evalúa: la estructura avanzada correcta con thead, tbody, th, td y el atributo colspan bien aplicado.

PREGUNTA 48:

Explica con tus palabras qué es la accesibilidad web y da tres ejemplos de cómo lograrla usando HTML. Se evalúa: el concepto de que la web debe ser usable por todas las personas, incluyendo personas con discapacidad, y tres ejemplos correctos de etiquetas o atributos (como alt, label o aria-label).

PREGUNTA 49:

Escribe en un bloque de código un formulario que use fieldset con su legend para agrupar los datos, y un input de texto con un datalist que sugiera tres ciudades al escribir. Se evalúa: las etiquetas fieldset y legend, el datalist con su id, el input con el atributo list que lo referencia y al menos tres opciones.

PREGUNTA 50:

Escribe en un bloque de código la estructura base de tu portafolio: un header con el logo y la navegación, un main con una section que tenga id="sobre-mi", y un footer con los derechos de autor. Se evalúa: el uso de etiquetas semánticas en el orden correcto, el atributo id en la sección y el contenido básico de cada parte.

Respuestas y soluciones

RESPUESTA 1:

Respuesta correcta: B) La última versión del lenguaje de marcado con el que se construyen todas las páginas web. HTML no es un lenguaje de programación porque no tiene lógica, variables ni condiciones: es un lenguaje de marcado, es decir, un sistema de etiquetas que describe la estructura y el contenido de una página. El 5 indica que es la versión más moderna, que agregó etiquetas semánticas, audio y video nativos, canvas y formularios inteligentes. Todas las páginas web del mundo, sin excepción, están construidas sobre HTML. La opción A es incorrecta porque los lenguajes de programación como JavaScript o Python sirven para dar lógica y comportamiento, no para estructurar. La opción C confunde HTML con un programa de diseño como Photoshop. La opción D confunde el lenguaje con el navegador, que es el programa que interpreta el HTML y lo muestra en pantalla.

RESPUESTA 2:

Respuesta correcta: C) <!DOCTYPE html>. Esta declaración debe ser la primera línea de todo documento HTML5, incluso antes de la etiqueta <html>. No es una etiqueta HTML como tal, sino una instrucción para el navegador que le dice: 'este documento usa el estándar HTML5, interprétalo en modo moderno'. Si la omite, el navegador entra en lo que se llama 'modo

peculiar' (quirks mode) y puede mostrar tu página de forma inconsistente, con medidas y estilos raros que varían entre navegadores. En el curso aprendiste que toda página empieza con DOCTYPE html, etiqueta html que envuelve todo, head con el título y body con el contenido. Las opciones A, B y D son etiquetas reales del documento, pero ninguna va primero: el orden correcto es DOCTYPE, luego html, y dentro de html van head y body.

RESPUESTA 3:

Respuesta correcta: B) <h1> y . La etiqueta <h1> define el título principal de la página: debe haber solo uno por página porque les dice a los buscadores y a los lectores de pantalla cuál es el tema central. Por otro lado, marca un texto como muy importante y los navegadores lo muestran en negrita. La diferencia clave con es el significado: solo pone negrita visual sin decir nada sobre importancia, mientras que comunica importancia a Google y a las tecnologías de asistencia. Lo mismo pasa con (énfasis, cursiva con significado) frente a <i> (cursiva solo visual). En el curso aprendiste a preferir siempre la versión con significado. Las demás opciones mezclan etiquetas que no corresponden: <p> es un párrafo, <title> va en el head y no se ve en la página, y <header> es una sección, no un título.

RESPUESTA 4:

Respuesta correcta: A) Google y Contacto. La etiqueta <a> (anchor, ancla) es la única que crea enlaces y el atributo href indica el destino. Para sitios externos se usa la URL completa con https:// para que el navegador sepa que debe salir de tu sitio. Para páginas internas se usa una ruta relativa como contacto.html, que el navegador completa con la dirección de tu propio sitio. La opción B es incorrecta porque <link> sirve para conectar recursos en el <head> (como hojas de estilo CSS), no para enlaces navegables, y además usa href, no src. La opción C inventa un atributo link que no existe. La opción D usa href como si fuera una etiqueta, lo cual es un error común de principiante: href siempre es un atributo dentro de <a>, nunca una etiqueta por sí sola.

RESPUESTA 5:

Respuesta correcta: C) alt. El atributo alt (texto alternativo) describe el contenido de la imagen con palabras. Es indispensable por tres razones: primero, accesibilidad, porque los lectores de pantalla lo leen en voz alta para personas con discapacidad visual; segundo, si la imagen no carga por un error o por conexión lenta, el navegador muestra ese texto en su lugar; tercero, ayuda al SEO porque Google no 've' imágenes pero sí lee el alt. La opción A (src) es la ruta del archivo de imagen: sin src no hay imagen, pero no tiene nada que ver con accesibilidad. La

opción B (width) solo controla el tamaño visual. La opción D (href) ni siquiera es un atributo de `<img>`: pertenece a los enlaces `<a>`. Regla de oro que viste en la clase 23: nunca publiques una imagen sin alt.

RESPUESTA 6:

Respuesta correcta: B) `ul` crea una lista con viñetas (no ordenada) y `ol` crea una lista numerada (ordenada). La etiqueta `<ul>` (unordered list) agrupa elementos donde el orden no importa: el navegador los muestra con viñetas. Es perfecta para menús de navegación o características de un producto. La etiqueta `<ol>` (ordered list) numera los elementos automáticamente (1, 2, 3...) y se usa cuando el orden sí importa, como los pasos de una receta o un ranking. En ambos casos cada elemento va dentro de `<li>` (list item). La opción A es incorrecta porque ninguna de las dos es exclusiva de enlaces o imágenes: dentro de un `<li>` puedes poner lo que quieras. La opción C es falsa: ambas son igual de modernas y se complementan. La opción D ignora la diferencia fundamental del orden.

RESPUESTA 7:

Respuesta correcta: C) `<th>`. La etiqueta `<th>` (table header) define una celda de encabezado: los navegadores la muestran en negrita y centrada por defecto para distinguirla visualmente de los datos. Se usa para los títulos de las columnas (como Nombre y Edad) o de las filas. La opción A (`<td>`, table data) es para las celdas con datos normales, no encabezados. La opción B (`<tr>`, table row) define una fila completa, no una celda. La opción D (`<thead>`) es un contenedor que agrupa las filas de encabezado de la tabla, pero las celdas dentro de él siguen siendo `<th>`. Un error común es creer que `<th>` solo puede vivir dentro de `<thead>`, pero en realidad puedes usar `<th>` en cualquier fila cuando esa celda sea un encabezado.

RESPUESTA 8:

Respuesta correcta: A) `div` es un contenedor de bloque y `span` es un contenedor en línea para fragmentos de texto. La etiqueta `<div>` es un elemento de bloque: ocupa todo el ancho disponible y siempre empieza en una línea nueva. Sirve para agrupar secciones grandes de la página, como una tarjeta o una columna. La etiqueta `<span>` es un elemento en línea: fluye dentro del texto sin romper la línea y sirve para marcar un fragmento pequeño, como una palabra a la que le quieres cambiar el color con CSS. En el curso los llamaste 'los cajones básicos para agrupar contenido'. La opción B es incorrecta porque en un `span` puedes poner texto y en un `div` casi cualquier cosa, incluyendo imágenes. La opción C es falsa: ninguna está obsoleta. La opción D ignora la diferencia fundamental entre bloque y línea, que es uno de los conceptos más importantes de HTML.

RESPUESTA 9:

Respuesta correcta: B) id es único en toda la página y class se puede usar en varios elementos. El atributo id funciona como la credencial de elector de un elemento: solo puede haber uno con ese nombre en todo el documento. Se usa para identificar un elemento específico, por ejemplo para que un `<label>` se conecte con su `<input>` o para crear enlaces internos a una sección. El atributo class funciona como un uniforme de equipo: muchos elementos pueden compartir la misma clase para recibir el mismo estilo con CSS. Si repites un id, el código puede comportarse de forma impredecible. La opción A invierte los conceptos. La opción C es falsa: tanto id como class se usan en CSS y en JavaScript. La opción D ignora la regla más importante de los atributos globales que viste en la clase 10.

RESPUESTA 10:

Respuesta correcta: C) `<button>`. La etiqueta `<button>` colocada dentro de un `<form>` crea el botón que envía los datos del formulario al servidor cuando el usuario le da clic. Puedes cambiar su texto simplemente escribiendo entre la apertura y el cierre, por ejemplo `<button>Enviar</button>`. La opción A (`<input type="text">`) crea un campo para escribir texto, no un botón. Las opciones B (`<send>`) y D (`<submit>`) son etiquetas inventadas que no existen en HTML: un error típico es suponer que existen porque suenan lógicas, pero en HTML5 el botón de envío es `<button>` o, alternativamente, `<input type="submit">`. Recuerda la estructura que viste en la clase 11: form envuelve todo, los input recogen los datos y button los envía.

RESPUESTA 11:

Respuesta correcta: B) `<input type="email">`. HTML5 incluye tipos de input inteligentes y `type="email"` es uno de los más útiles: el navegador revisa automáticamente que lo escrito tenga formato de correo (algo como `usuario@dominio.com`) y si no lo tiene, bloquea el envío y muestra un mensaje de error sin que tú escribas ni una línea de JavaScript. Además, en celulares abre el teclado con la arroba y el punto a la mano, lo que mejora la experiencia del usuario. La opción A (`type="text"`) acepta cualquier texto sin validar nada. Las opciones C (`type="mail"`) y D (`type="correo"`) no existen: son inventos que parecen lógicos pero el navegador los trataría como un simple campo de texto. En la clase 12 viste otros inputs inteligentes como `password`, `date` y `color`, cada uno con su comportamiento especial.

RESPUESTA 12:

Respuesta correcta: B) `required`. El atributo `required` convierte un campo en obligatorio: si el usuario intenta enviar el formulario sin llenarlo, el navegador lo detiene y muestra un aviso automáticamente. Es parte de la validación nativa de HTML5, que funciona sin JavaScript, sin

librerías y sin escribir código extra. La opción A (placeholder) solo muestra un texto de ayuda gris dentro del campo que desaparece al escribir: no obliga a nada. La opción C (value) define un valor precargado en el campo, no una obligación. La opción D (name) le pone nombre al campo para identificarlo cuando los datos viajan al servidor, pero tampoco válida. La validación nativa que viste en la clase 13 (required, minlength, type="email") te ahorra mucho trabajo y mejora la experiencia porque el navegador se encarga de todo.

RESPUESTA 13:

Respuesta correcta: C) <div>. Una etiqueta es semántica cuando su nombre describe el significado del contenido que envuelve: <article> dice 'esto es un artículo independiente', <nav> dice 'esto es navegación' y <footer> dice 'esto es el pie de página'. En cambio, <div> no dice nada sobre su contenido: es un contenedor genérico, una caja vacía que sirve para agrupar pero no comunica significado ni a Google ni a los lectores de pantalla. Por eso en la clase 24 aprendiste la regla 'semántica antes que div': usa las etiquetas con significado siempre que puedas y reserva el div para cuando ninguna etiqueta semántica encaje. Las opciones A, B y D son todas semánticas y forman parte del vocabulario que HTML5 agregó justamente para tener código más limpio y mejor posicionado.

RESPUESTA 14:

Respuesta correcta: B) Para que la página se vea bien en celulares ajustando el ancho al dispositivo. La meta etiqueta viewport le dice al navegador móvil: 'ajusta el ancho de la página al ancho de la pantalla del dispositivo y empieza con zoom al 100 por ciento'. Sin ella, el celular muestra la página como si fuera de escritorio, con todo en miniatura y obligando al usuario a hacer zoom con los dedos: una experiencia terrible. Su sintaxis típica es <meta name="viewport" content="width=device-width, initial-scale=1.0">. La opción A es incorrecta porque el color se maneja con CSS, no con meta tags. La opción C confunde viewport con la meta description, que sí aparece en Google. La opción D es falsa: el viewport no acelera nada, solo controla cómo se dimensiona la página. Hoy más de la mitad del tráfico web es móvil, así que este meta tag es obligatorio.

RESPUESTA 15:

Respuesta correcta: B) controls. El atributo controls le dice al navegador que muestre la barra de reproducción: botón de play y pausa, control de volumen, barra de progreso y pantalla completa. Sin controls, el audio o el video se cargan pero el usuario no tiene forma de reproducirlos (a menos que uses autoplay o JavaScript). La opción A (autoplay) inicia la reproducción automáticamente al cargar la página, lo cual suele ser molesto y muchos navegadores lo bloquean si tiene sonido. Las opciones C (play) y D (media) no existen como

atributos: son palabras que suenan relacionadas pero el navegador las ignoraría. En la clase 16 aprendiste que con `<audio controls src="musica.mp3">` y `<video controls src="video.mp4">` tienes multimedia nativa sin plugins, algo que antes de HTML5 requería programas externos.

RESPUESTA 16:

Respuesta correcta: B) JavaScript, porque canvas es un lienzo vacío que se dibuja con código. La etiqueta `<canvas>` solo crea un espacio rectangular vacío en la página, como un pizarrón en blanco: por sí sola no dibuja absolutamente nada. Para trazar líneas, círculos, texto o animaciones necesitas JavaScript, que accede al canvas por su id y usa su contexto de dibujo. Es la base para juegos, gráficas y animaciones hechas con código. La opción A es el error más común: creer que la etiqueta hace magia sola. La opción C inventa una etiqueta `<draw>` que no existe. La opción D confunde canvas con `<img>`: una imagen muestra un archivo, canvas es un área de dibujo programable. En la clase 19 lo definiste perfecto: 'Canvas es tu pizarra para dibujar con código'.

RESPUESTA 17:

Respuesta correcta: B) Los gráficos SVG son vectoriales y no se pixelan al agrandarlos. SVG (gráficos vectoriales escalables) dibuja formas con matemáticas: círculos, rectángulos y rutas definidos por coordenadas. Como no está hecho de píxeles, puedes agrandarlos al tamaño de un espectacular y se seguirá viendo nítido: por eso es perfecto para logotipos e iconos. Canvas, en cambio, es un mapa de píxeles: si lo escalas más allá de su tamaño original, se pixela. La opción A es incorrecta porque para videojuegos con muchos gráficos por segundo Canvas suele rendir mejor. La opción C invierte la realidad: Canvas necesita JavaScript y SVG se escribe directo en el HTML. La opción D es falsa: SVG es ideal para ilustraciones y logos, mientras que las fotografías (hechas de píxeles) siguen siendo trabajo de formatos como JPG.

RESPUESTA 18:

Respuesta correcta: B) `<`. Las entidades HTML son códigos que representan caracteres especiales: empiezan con `&` y terminan con `;`. La entidad `<` significa 'less than' (menor que) y muestra el símbolo `<` como texto visible sin que el navegador lo confunda con el inicio de una etiqueta. Esto es esencial cuando quieres mostrar código como texto en tu página, por ejemplo para enseñar HTML. Su pareja es `>` (greater than) para el símbolo `>`. La opción A (` `) crea un espacio en blanco que el navegador no colapsa. La opción C (`&`) muestra el símbolo `&` mismo. La opción D (`©`) muestra el símbolo de derechos de autor. En la clase 22 aprendiste que sin estas entidades sería imposible escribir un tutorial de HTML dentro de una página HTML.

RESPUESTA 19:

Respuesta correcta: B) Para que los lectores de pantalla describan la imagen a personas con discapacidad visual. El atributo alt tiene una misión de accesibilidad: los lectores de pantalla (programas que usan las personas ciegas o con baja visión) leen en voz alta el texto del alt para describir lo que muestra la imagen. Un buen alt dice lo importante de la imagen ('perro labrador jugando en el parque'), no 'imagen' ni 'foto1'. Además, si la imagen no carga, el navegador muestra ese texto, y Google lo usa para entender de qué trata la imagen y posicionarla en su buscador de imágenes. La opción A es incorrecta porque el tamaño se controla con width y height. La opción C es falsa: el formato lo define el archivo, no el atributo. La opción D ignora todo lo que viste en la clase 23: la web debe ser para todos.

RESPUESTA 20:

Respuesta correcta: B) <marquee>. La etiqueta <marquee> creaba textos en movimiento y fue popular en los años 90, pero hoy está obsoleta: los navegadores modernos pueden dejar de soportarla en cualquier momento y su efecto se logra mejor con CSS o animaciones controladas. En la clase 26 aprendiste la lista negra: no uses <center>, ni <marquee>. Todas mezclaban presentación con estructura, y la filosofía de HTML5 es clara: HTML es estructura, CSS es estilo. Las opciones A (<section>), C (<video>) y D (<article>) son etiquetas modernas y totalmente recomendadas: section y article son semánticas y video es multimedia nativa. Si en un examen o en un trabajo ves center, font o marquee, ya sabes que es código viejo que hay que modernizar.

RESPUESTA 21:

F. La declaración <!DOCTYPE html> debe ir siempre como primera línea del documento. Aunque los navegadores intentan mostrar la página sin ella, entran en 'modo peculiar' y pueden renderizar medidas, márgenes y estilos de forma inconsistente entre navegadores. Es una línea que no cuesta nada escribir y garantiza que tu página se interprete con el estándar moderno. En el curso aprendiste que toda página empieza con DOCTYPE: no es decoración, es la base. Piensa en el DOCTYPE como el acta de nacimiento de tu página: declara bajo qué reglas nació. En el trabajo real, los validadores de código marcan su ausencia como error, y depurar diseños rotos por el modo peculiar puede costarte horas. Los profesionales lo escriben en automático, sin pensarlo.

RESPUESTA 22:

F. La etiqueta
 es una etiqueta vacía: no envuelve contenido, solo inserta un salto de línea donde aparece. Las etiquetas vacías como
, <hr> e no llevan etiqueta de cierre en

HTML5. Escribir `</br>` es un error de sintaxis que los navegadores toleran pero que un buen desarrollador nunca comete. Recuerda la diferencia que viste en la clase 4: `<p>` sí necesita cierre porque envuelve un párrafo, pero `<br>` y `<hr>` actúan solas. En HTML5 hay toda una familia de etiquetas vacías: `br`, `hr`, `img`, `input`, `meta`, `link` y `source`, entre otras, y ninguna lleva cierre. Dato curioso: en XHTML se escribían con una diagonal al final (`<br/>`), pero en HTML5 la forma simple (`<br>`) es la recomendada.

RESPUESTA 23:

F. Publicar una imagen sin `alt` es una mala práctica grave por tres motivos: las personas con discapacidad visual no sabrán qué muestra la imagen porque su lector de pantalla no tendrá nada que leer; si la imagen falla al cargar, el usuario verá un recuadro vacío sin explicación; y pierdes posicionamiento porque Google usa el `alt` para entender y clasificar tus imágenes. En la clase 23 quedó claro: el `alt` no es opcional, es parte de hacer una web profesional y accesible. En proyectos reales, omitir el `alt` se considera un defecto de calidad: afecta a usuarios reales, daña el posicionamiento de tus imágenes y en algunos países hasta tiene implicaciones legales de accesibilidad. Acostúmbrate a escribirlo al mismo tiempo que el `src`, no después: así nunca se te olvida.

RESPUESTA 24:

F. La etiqueta `<li>` solo tiene sentido dentro de una lista (`<ul>`, `<ol>` o `<menu>`): representa un elemento de esa lista. Usarla suelta, fuera de una lista, es código inválido según el estándar HTML5, aunque el navegador intente mostrarla. Es como poner un capítulo fuera de un libro: el capítulo solo existe como parte del libro. La estructura correcta que viste en la clase 7 siempre es lista primero y elementos dentro: `<ul>` abre, `<li>` define cada item y `</ul>` cierra. Los validadores de HTML marcan el `<li>` huérfano como error, y en equipos de trabajo el código que no válida se regresa para corregir. Además, los lectores de pantalla anuncian las listas ('lista con 4 elementos'), así que un `<li>` suelto rompe esa experiencia. La regla es simple: sin lista, no hay `<li>`.

RESPUESTA 25:

F. La etiqueta `<th>` define una celda de encabezado y puede usarse en cualquier fila de la tabla, no solo dentro de `<thead>`. Por ejemplo, es común usar `<th>` como primera celda de cada fila cuando esa columna funciona como encabezado lateral (como los días de la semana en un horario). Lo que hace `<thead>` es agrupar las filas de encabezado de la tabla como bloque, pero las celdas de encabezado individuales siguen siendo `<th>` estén donde estén. Confundir el contenedor con la celda es un error frecuente. Un ejemplo clásico: en un horario semanal, la primera columna con los días (lunes, martes...) usa `<th>` en cada fila del cuerpo

porque son encabezados de fila. El `thead` agrupa filas completas de encabezado; el `th` marca celdas de encabezado donde sea que estén. Son dos conceptos distintos que trabajan juntos.

RESPUESTA 26:

F. Es exactamente al revés: el `id` debe ser único en toda la página y el `class` está diseñado para repetirse en varios elementos. El `id` identifica a un elemento específico (como una credencial única) y sirve para conectarlo con un `<label>`, para enlaces internos o para seleccionarlo con JavaScript. El `class` agrupa elementos que comparten características (como un uniforme de equipo) para darles el mismo estilo con CSS. Si repites un `id`, el navegador y JavaScript pueden comportarse de forma impredecible. Esta fue la lección central de la clase 10. Si repites un `id`, el CSS puede aplicar estilos de forma inconsistente y JavaScript solo encontrará el primer elemento con ese `id`, ignorando los demás: son errores difíciles de rastrear. Por eso la convención es clara: `id` para lo único (como `#menú-principal`) y `class` para lo repetible (como `boton`).

RESPUESTA 27:

V. El `input type="password"` está diseñado para campos de contraseña: todo lo que el usuario escribe se muestra como puntos o asteriscos para que nadie pueda leerlo por encima del hombro. Ojo con un detalle importante que debes entender: esto solo oculta el texto en pantalla, no lo encripta ni lo protege al enviarse; la seguridad real la da el servidor con conexiones cifradas. Aun así, para la experiencia del usuario es el tipo correcto siempre que pidas una contraseña, tal como viste en la clase 12 junto a `email`, `date` y `color`. El navegador también permite alternar entre mostrar y ocultar la contraseña cambiando el `type` entre `password` y `text` con JavaScript: es el famoso ojito de 'mostrar contraseña'. Pero el tipo base en el HTML siempre debe ser `password`, porque es el que protege lo que se ve en pantalla desde el inicio.

RESPUESTA 28:

V. El atributo `required` es parte de la validación nativa de HTML5: el propio navegador impide el envío del formulario y muestra un mensaje de error si el campo está vacío, sin que escribas una sola línea de JavaScript. Esto funciona en todos los navegadores modernos y es la forma más rápida y accesible de validar. En la clase 13 viste que se combina con otros atributos como `minlength` y `type="email"` para validaciones más completas. Eso sí, recuerda que la validación del navegador es una primera capa de ayuda al usuario: un sitio profesional también válida en el servidor. Esta validación mejora mucho la experiencia: el usuario recibe el aviso al instante, en su idioma y sin recargar la página. Existe el atributo `novalidate` para desactivarla cuando necesitas una validación personalizada con JavaScript, pero por defecto conviene dejarla activa.

RESPUESTA 29:

F. Aunque visualmente una `<section>` sin estilos se ve igual que un `<div>`, no son lo mismo: `<section>` es semántica y comunica que su contenido forma una sección temática del documento (como 'sobre mí' o 'servicios'), mientras que `<div>` es un contenedor genérico sin significado. Esa diferencia la entienden Google y los lectores de pantalla, y por eso el código semántico posiciona mejor y es más accesible. La regla de la clase 24 es clara: semántica antes que div. Usa `<div>` solo cuando ninguna etiqueta semántica describa bien tu contenido. Google premia el contenido bien estructurado y los lectores de pantalla permiten saltar por regiones ('ir al contenido principal', 'ir a la navegación'): con puros divs esa navegación se pierde. El div sigue siendo útil para envoltorios de diseño sin significado, pero el contenido con sentido merece su etiqueta semántica. Esa es la mentalidad profesional.

RESPUESTA 30:

F. La meta etiqueta `viewport` existe precisamente para los celulares: le dice al navegador móvil que ajuste el ancho de la página al ancho del dispositivo. En computadoras de escritorio no tiene efecto relevante porque la pantalla ya es ancha. Sin `viewport`, el celular muestra tu página como una versión miniatura de escritorio y el usuario tiene que hacer zoom con los dedos para leer. Con más de la mitad del tráfico web viniendo de móviles, omitir el `viewport` es uno de los errores más graves que puedes cometer, como viste en la clase 15. Google además usa indexación `mobile-first`: evalúa primero cómo se ve tu página en celular para posicionarla. En la práctica, este meta tag se escribe una vez en la plantilla base y se hereda en todo el sitio: tres segundos que te ahorran un desastre.

RESPUESTA 31:

V. La etiqueta `<iframe>` incrusta una página web dentro de otra, y YouTube ofrece justamente ese formato: en cada video hay un botón de compartir que te da un código `<iframe>` listo para

pegar. Al ponerlo en tu página, el video de YouTube se reproduce dentro de tu sitio sin que tengas que alojar el archivo. En la clase 18 viste que lo mismo funciona con mapas de Google y otros contenidos. Un consejo profesional: usa el atributo title en el iframe para accesibilidad y evita abusar de ellos porque cada iframe carga una página completa y puede alentar tu sitio. Para hacerlo aún mejor, describe el contenido en el title ('Video: curso de HTML5') para los lectores de pantalla y considera el atributo loading="lazy" para que el iframe cargue solo cuando el usuario se acerque a él. Pequeños detalles que hacen tu página más rápida y accesible.

RESPUESTA 32:

F. La etiqueta <canvas> por sí sola es un lienzo vacío: crea el espacio rectangular en la página pero no dibuja nada. Todo el dibujo (líneas, círculos, texto, animaciones) se hace con JavaScript, que accede al canvas mediante su id y usa su contexto de dibujo 2D. Sin JavaScript, el canvas es invisible e inútil. Esta es la diferencia clave con SVG, que sí dibuja formas directo desde el HTML. En la clase 19 lo resumiste bien: canvas es tu pizarra para dibujar con código, y el código es JavaScript. Con JavaScript obtienes el contexto de dibujo y desde ahí trazas todo: rectángulos, círculos, texto e incluso animaciones cuadro por cuadro. Por eso canvas es la base de los juegos web y las gráficas interactivas. Sin JavaScript es solo una caja vacía reservando espacio: existe en el HTML, pero no hace nada.

RESPUESTA 33:

V. La estructura de una lista de definición siempre es <dl> como contenedor (definition list), <dt> para cada término (definition term) y <dd> para su definición (definition description). Los <dt> y <dd> huérfanos, fuera de un <dl>, son código inválido porque el término y la definición solo tienen sentido como parte de la lista. En la clase 21 la viste como la herramienta perfecta para glosarios: por ejemplo, <dt>HTML</dt> seguido de <dd>Lenguaje de marcado para crear páginas web</dd>. Puedes tener varios <dt> o varios <dd> seguidos dentro del mismo <dl> sin problema. Esta estructura es tan flexible que sirve para glosarios, preguntas frecuentes y hasta diálogos: varios términos pueden compartir una definición y viceversa. Con un poco de CSS, una lista de definición se convierte en una sección de preguntas frecuentes elegante sin necesidad de nada más. Es una etiqueta subestimada que los profesionales usan mucho.

RESPUESTA 34:

V. Comentar el código con <!-- tu comentario --> es una de las cuatro buenas prácticas de la clase 24, junto con indentar, usar minúsculas y preferir semántica. Los comentarios no se ven en la página pero explican el código a quien lo lea después: tu yo del futuro, un compañero de trabajo o un cliente que revise el proyecto. Un buen comentario dice el porqué de algo no

obvio ('aquí empieza la sección de contacto'), no lo evidente. El código se escribe una vez pero se lee muchas veces, así que comentar es una inversión que siempre paga. Un buen comentario explica el porqué, no el qué: '

respuesta tiene este orden y todos los cierres, está perfecta. Un tip de profesional: memoriza esta plantilla como una canción, porque la vas a escribir cientos de veces en tu carrera. Muchos editores la generan con un atajo de teclado, pero en un examen debes saberla de memoria: es tu credencial de desarrollador web.

RESPUESTA 37:

Puntos clave que debe contener una buena respuesta: HTML5 es la última versión del lenguaje de marcado (no un lenguaje de programación) con el que se construyen todas las páginas web; sirve para definir la estructura y el contenido (títulos, párrafos, imágenes, enlaces, formularios) usando etiquetas; incluye novedades como etiquetas semánticas, audio y video nativos, canvas y formularios inteligentes; y el siguiente paso después de dominarlo es aprender CSS3, que le da estilo, colores y diseño a esa estructura. Una respuesta completa explica que HTML es el esqueleto de la web: sin él no hay página, y que CSS3 es el paso natural porque separa la presentación de la estructura, tal como viste en las clases 1 y 30 del curso. Un tip: cuando alguien te pregunte a qué te dedicas, esta respuesta es tu presentación. Decir 'construyo páginas web con HTML' suena simple, pero detrás hay un oficio completo: estructura, semántica, accesibilidad y formularios que sí funcionan.

RESPUESTA 38:

Código correcto:

```
<p>Este es un texto normal con una palabra en <strong>negrita</strong> y otra en  
<em>cursiva</em>.<br>Esta línea empieza después del salto.</p>  
<hr>
```

El `<p>` envuelve todo el párrafo y es obligatorio cerrarlo. Dentro de él, `<strong>` marca la palabra importante (el navegador la muestra en negrita) y `<em>` marca el énfasis (cursiva con significado). El `<br>` es una etiqueta vacía que corta la línea exactamente donde aparece, sin crear un párrafo nuevo. El `<hr>` va fuera del párrafo porque es un separador temático entre bloques, no parte del texto, y también es etiqueta vacía sin cierre. Si tu código usa cada etiqueta en su lugar y cierra las que necesitan cierre, la respuesta es correcta. Recuerda: nunca uses `<br>` para separar párrafos enteros; para eso existen varios `<p>`. Un tip de profesional: abusar del `<br>` para crear espacios es una marca típica del código de principiante. Si necesitas separar bloques con aire, eso lo hará CSS con márgenes. El `<br>` es para saltos dentro del mismo bloque de texto, como en una dirección postal o un poema.

RESPUESTA 39:

Código correcto:

```
<a href="https://lucasegaacademy.com" target="_blank">Lucas EGA Academy</a>  
<a href="contacto.html">Contacto</a>
```

El primer enlace es externo: lleva la URL completa con https:// para salir de tu sitio, y target="_blank" para que abra en una pestaña nueva sin sacar al usuario de tu página. El segundo es interno: usa la ruta relativa contacto.html y abre en la misma pestaña, que es el comportamiento esperado al navegar dentro de un sitio. El error más común es olvidar el https:// en el externo (el navegador lo trataría como página interna) o poner target="_blank" en enlaces internos, lo que llena de pestañas al usuario sin necesidad. La etiqueta siempre es <a> y el destino siempre va en href: eso no cambia nunca. Un tip de profesional: en el enlace externo puedes agregar rel="noopener" junto a target="_blank" por seguridad, para que la página nueva no tenga acceso a la tuya. No era obligatorio en esta pregunta, pero en proyectos reales es una buena costumbre que te hará ver como alguien que sabe.

RESPUESTA 40:

Código correcto:

```

```

La etiqueta es vacía: no lleva cierre. El atributo src indica la ruta del archivo de imagen y es obligatorio para que algo se muestre. El atributo alt lleva una descripción real de la imagen (no 'imagen' ni el nombre del archivo): es tu compromiso con la accesibilidad y el SEO. Los atributos width y height definen el tamaño en píxeles y además ayudan a que la página no 'salte' mientras carga la imagen. El orden de los atributos no importa, pero los cuatro deben estar presentes. Si tu respuesta incluye src, un alt descriptivo, width="300" y height="200", está completa. Un tip de profesional: guarda tus imágenes optimizadas para web (con el ancho real de uso) antes de subirlas. Una foto pesada frena tu página aunque el width diga 300. El HTML pone la imagen, pero el rendimiento lo cuidas tú desde el archivo.

RESPUESTA 41:

Código correcto:


```
<table>
<tr>
<th>Nombre</th>
<th>Edad</th>
</tr>
<tr>
<td>Lucas</td>
<td>25</td>
</tr>
<tr>
<td>Ana</td>
<td>30</td>
</tr>
</table>
```

La etiqueta `<table>` envuelve toda la tabla. Cada `<tr>` (table row) define una fila. La primera fila usa `<th>` (table header) para los encabezados Nombre y Edad, que el navegador muestra en negrita y centrados. Las filas de datos usan `<td>` (table data) para cada celda. Todas las etiquetas necesitan su cierre y la anidación es estricta: las celdas van dentro de las filas y las filas dentro de la tabla. Un error típico es mezclar `<th>` y `<td>` o dejar etiquetas sin cerrar, lo que rompe la tabla. Con dos columnas de encabezado y dos filas de datos como pide la pregunta, tu respuesta está completa. Un tip de profesional: hoy las tablas se usan solo para datos tabulares, nunca para maquetar el diseño de la página (eso se hacía en los años 90 y hoy lo hace CSS). Si tus datos no tienen filas y columnas con sentido, probablemente no necesitas una tabla.

RESPUESTA 42:

Puntos clave que debe contener una buena respuesta: `<div>` es un contenedor de bloque que ocupa todo el ancho disponible y empieza en una línea nueva, ideal para agrupar secciones grandes como tarjetas, columnas o bloques de contenido; `<span>` es un contenedor en línea que fluye dentro del texto sin romper la línea, ideal para marcar fragmentos pequeños como una palabra o frase. Ejemplo de `div`: envolver una tarjeta de producto con su imagen, título y precio. Ejemplo de `span`: pintar de otro color una palabra dentro de un párrafo. Una buena respuesta deja claro que la diferencia fundamental es bloque contra en línea, y que esa diferencia determina cuándo usar cada uno.

RESPUESTA 43:

Código correcto:

```
<form>
<input type="text" placeholder="Escribe tu nombre">
<input type="email" required placeholder="tu@correo.com">
<button>Enviar</button>
</form>
```

El `<form>` envuelve todos los campos y es el que se encargará de enviar los datos. El primer input es `type="text"` para el nombre, con placeholder que muestra una ayuda gris que desaparece al escribir. El segundo input es `type="email"` para que el navegador valide el formato de correo, con `required` para hacerlo obligatorio: si está vacío o mal escrito, el navegador bloquea el envío sin necesidad de JavaScript. El `<button>` dentro del form envía los datos al hacer clic. Si tu formulario tiene estos tres elementos con sus atributos correctos, la respuesta está completa. Recuerda: placeholder ayuda pero no válida, required obliga pero no ayuda; se usan juntos. Un tip de profesional: el atributo `name` en cada input es lo que permite que los datos viajen identificados al servidor. Sin `name`, el campo se ve bien pero sus datos llegan huérfanos. En esta pregunta no se pedía, pero en un formulario real ningún input viaja sin su `name`.

RESPUESTA 44:

Puntos clave que debe contener una buena respuesta: el HTML semántico usa etiquetas cuyo nombre describe el significado del contenido (no solo su apariencia); al menos cuatro etiquetas correctas entre `header`, `nav`, `main`, `section`, `article`, `aside`, `footer`, `figure` o `time`; y a Google le gusta porque su robot entiende mejor de qué trata cada parte de la página (dónde está el contenido principal, qué es navegación, qué es un artículo independiente), lo que mejora el posicionamiento en los resultados de búsqueda (SEO). Una respuesta completa contrasta con el `<div>` genérico que no comunica nada, y menciona que la semántica también ayuda a los lectores de pantalla, así que ganas en SEO y en accesibilidad al mismo tiempo. Un tip: cuando conviertas un diseño a HTML, primero escribe solo las etiquetas semánticas sin pensar en cómo se ve. Si tu página se entiende leyendo solo el HTML, tu semántica es correcta. El diseño viene después, con CSS.

RESPUESTA 45:

Código correcto:

```
<meta charset="UTF-8">
<meta name="description" content="Descripción breve de mi página">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Estos tres van dentro del <head> y cada uno cumple una misión: charset="UTF-8" permite mostrar correctamente acentos, eñes y caracteres especiales; name="description" da el resumen que Google muestra debajo de tu título en los resultados de búsqueda y mejora tu SEO; name="viewport" ajusta la página al ancho del celular para que se vea bien en móviles. Son meta tags, así que son etiquetas vacías sin cierre y usan atributos, no contenido entre etiquetas. Si tu respuesta incluye los tres con su sintaxis correcta, está perfecta. Sin ellos tu página tendría acentos rotos, mala descripción en Google y se vería en miniatura en el celular. Un tip de profesional: la description ideal mide entre 150 y 160 caracteres y resume con gancho lo que ofrece la página, porque es tu anuncio gratis en Google. Escríbela pensando en la persona que va a decidir si hace clic o no.

RESPUESTA 46:

Código correcto:

```
<video controls>
<source src="video.mp4" type="video/mp4">
Tu navegador no soporta video HTML5.
</video>
```

El atributo controls muestra la barra de reproducción (play, pausa, volumen, pantalla completa): sin él, el usuario no podría reproducir el video. La etiqueta <source> dentro de video indica el archivo y su tipo; también es válido usar el atributo src directo en <video>. El texto entre las etiquetas es el mensaje de respaldo: solo se muestra si el navegador no soporta video HTML5, así que nunca dejes ese espacio vacío. Esta es la multimedia nativa que viste en la clase 16: antes de HTML5 necesitabas plugins externos, ahora el navegador lo hace todo. Si tu código tiene video, controls, la fuente y el mensaje de respaldo, la respuesta está completa. Un tip de profesional: ofrece tu video en más de un formato (mp4 y webm) con varias etiquetas source para máxima compatibilidad entre navegadores. Detalles así separan al aficionado del profesional.

RESPUESTA 47:

Código correcto:

```
<table>
<thead>
<tr>
<th colspan="2">Datos del estudiante</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lucas</td>
<td>25</td>
</tr>
<tr>
<td>Ana</td>
<td>30</td>
</tr>
</tbody>
</table>
```

Esta es la estructura profesional de una tabla: `<thead>` agrupa las filas de encabezado y `<tbody>` agrupa las filas de datos, lo que ayuda a los lectores de pantalla y permite dar estilos distintos a cada parte. El atributo `colspan="2"` en el `<th>` hace que esa celda abarque dos columnas, fusionándolas en una sola celda ancha para el título general. Existe también `rowspan` para fusionar filas. Si tu tabla usa `thead` y `tbody` correctamente y el `colspan` está aplicado al `th` con el valor 2, la respuesta es correcta. Recuerda: `colspan` cuenta columnas hacia la derecha, `rowspan` cuenta filas hacia abajo. Un tip de profesional: dibuja tu tabla en papel antes de escribir el código cuando uses `colspan` o `rowspan`. Marca qué celdas se fusionan y cuenta bien: una combinación mal calculada descuadra toda la tabla y es de los errores más tardados de depurar a simple vista.

RESPUESTA 48:

Puntos clave que debe contener una buena respuesta: la accesibilidad web significa que todas las personas, incluyendo personas con discapacidad visual, auditiva o motriz, puedan usar tu página sin barreras; y tres ejemplos correctos entre: atributo `alt` descriptivo en las imágenes para lectores de pantalla, etiqueta `<label>` asociada a cada `input` con el atributo `for` para que se sepa qué dato pide cada campo, atributo `aria-label` para dar nombre a elementos sin texto visible (como un botón de menú hamburguesa), subtítulos o texto alternativo en videos, y contraste y tamaños legibles. Una buena respuesta transmite la idea central de la clase 23: la web es para todos, y lograrlo no requiere trucos complicados, solo usar bien las herramientas que HTML ya te da. Un tip: prueba tu página solo con el teclado (tecla Tab para moverte) y con

un lector de pantalla gratuito. Lo que descubras te abrirá los ojos: la accesibilidad deja de ser teoría y se vuelve algo que sientes. Los mejores desarrolladores prueban así sus proyectos.

RESPUESTA 49:

Código correcto:

```
<form>
<fieldset>
<legend>Datos de envío</legend>
<input type="text" placeholder="Nombre">
<input list="ciudades" placeholder="Tu ciudad">
<datalist id="ciudades">
<option value="Ciudad de México">
<option value="Guadalajara">
<option value="Monterrey">
</datalist>
</fieldset>
<button>Enviar</button>
</form>
```

El `<fieldset>` agrupa campos relacionados con un borde visual y el `<legend>` le pone título a ese grupo ('Datos de envío'), lo que ordena formularios largos y ayuda a los lectores de pantalla. El `<datalist>` con su `id` ofrece sugerencias mientras el usuario escribe: el `<input>` se conecta a la lista mediante el atributo `list="ciudades"` (debe coincidir exactamente con el `id`). Cada `<option>` aporta una sugerencia. La diferencia con un `<select>` es que el `datalist` sugiere pero permite escribir algo distinto, mientras que el `select` obliga a elegir. Si tu código tiene `fieldset`, `legend`, `input` con `list`, `datalist` con `id` coincidente y tres opciones, está completo. Un tip de profesional: el `datalist` es perfecto para campos como ciudad o país, donde quieres ayudar sin limitar. Si la lista debe ser cerrada y obligatoria (como el método de pago), usa `select` en su lugar. Elegir el control correcto es parte del oficio.

RESPUESTA 50:

Código correcto:

```
<header>
<p>Mi logo</p>
<nav>
<a href="#sobre-mi">Sobre mí</a>
<a href="#contacto">Contacto</a>
</nav>
</header>
<main>
<section id="sobre-mi">
<h2>Sobre mí</h2>
<p>Soy desarrollador web.</p>
</section>
</main>
<footer>
<p>Derechos reservados 2026</p>
</footer>
```

Este es el esqueleto del proyecto final del curso: <header> arriba con el logo y el <nav> de navegación, <main> envolviendo el contenido principal (solo debe haber un main por página), la <section> con id="sobre-mi" que permite enlazar directo a esa parte con #sobre-mi, y el <footer> con los derechos. Todo con etiquetas semánticas en el orden lógico en que aparecen en la página. El id en la sección es clave porque conecta con el enlace del nav. Si tu estructura usa estas etiquetas semánticas en orden, incluye el id correcto y cada parte tiene su contenido básico, tu portafolio tiene la base profesional que piden las clases 27 a 29. ¡Felicidades, ya eres frontend! Un tip: este esqueleto es solo el inicio. Ahora llénalo con tu información real, agrega tus proyectos con imágenes y enlaces, publícalo gratis y compártelo. Un portafolio en línea vale más que un currículum: es la prueba viva de lo que sabes hacer.

Colibrí Editorial EGA

Curso Completo de HTML5

Curso Completo de HTML5 es el libro oficial del curso gratuito de Lucas EGA Academy. En 10 unidades cubre las 30 clases del curso en video: desde la estructura básica de un documento hasta formularios profesionales, multimedia, canvas, SVG y accesibilidad. Cada unidad explica con analogías, errores comunes y buenas prácticas, e incluye ejemplos paso a paso comentados línea por línea, prácticas con soluciones explicadas y un examen final de 50 preguntas con respuestas.

lucasegaacademy.com