

Unidad 2:

Fundamentos de

HTML5

HTML (HyperText Markup Language) es el **esqueleto** de cualquier sitio web. No "programa" nada por sí mismo; **describe y estructura** el contenido que luego el navegador interpreta para dibujar la página.

Sesión 3 – Introducción, estructura básica y etiquetas esenciales

HTML (HyperText Markup Language) es el **esqueleto** de cualquier sitio web. No "programa" nada por sí mismo; **describe y estructura** el contenido que luego el navegador interpreta para dibujar la página. Por eso decimos que es un **lenguaje de marcado**, no un lenguaje de programación: define **qué hay** y **cómo se organiza**, mientras que CSS decide **cómo se ve** y JavaScript **qué hace**.

La **estructura básica** de un documento moderno comienza con `<!DOCTYPE html>`, que indica al navegador que use el **modo estándar de HTML5**. A continuación, todo el documento se envuelve en `<html>` e idealmente se declara el idioma del contenido con lang (ej. `<html lang="es">`) para mejorar accesibilidad, SEO y renderizado de tipografías. Dentro de `<html>` hay dos grandes zonas:

- **<head>**: contiene **metadatos** que no son visibles como contenido principal, pero sí son esenciales para el navegador, buscadores y dispositivos. Aquí van la **codificación de caracteres** (`<meta charset="UTF-8">`), el **título del documento** (`<title>...</title>`), la **meta viewport** para móviles (`<meta name="viewport" content="width=device-width, initial-scale=1">`), enlaces a hojas de estilo (`<link rel="stylesheet" href="styles.css">`) o iconos.
- **<body>**: contiene **todo el contenido visible** (y audible) para la persona usuaria: encabezados, párrafos, imágenes, listas, enlaces, tablas, formularios, secciones semánticas y cualquier elemento que componga la página.

En HTML todo son **elementos** que abren y cierran (por ejemplo `<p>...</p>`), salvo casos particulares como `` o `
` que son *vacíos*. Es clave comprender la **anidación** (un elemento dentro de otro) y mantener una **indentación limpia** para leer y mantener el código con facilidad.

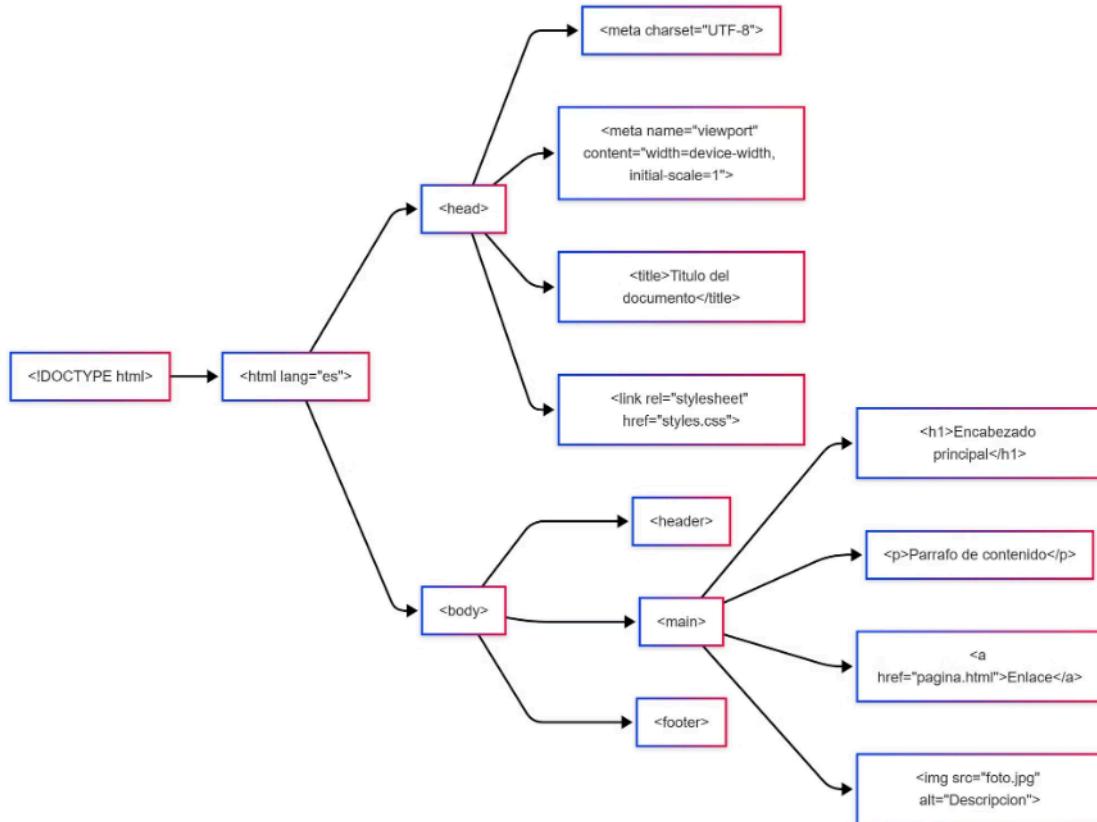
Para contenido "esencial", conviene dominar desde el principio estas etiquetas:

- **Encabezados** `<h1>...<h6>` (jerarquía de títulos, un solo `<h1>` por página),
- **Texto** `<p>`, énfasis con `` y ``,
- **Listas** ``, ``, ``,
- **Enlaces** ``,
- **Imágenes** `` (el atributo alt es obligatorio por accesibilidad),
- **Bloques semánticos básicos** `<header>`, `<main>`, `<footer>` (los veremos en detalle más adelante en la unidad).

En resumen: HTML5 aporta un marco **sencillo pero riguroso**. Si respetas su estructura mínima, usas etiquetas **semánticas** (la etiqueta correcta para el tipo de contenido) y escribes un código ordenado, ya estás construyendo sobre cimientos profesionales.

Esquema Visual

A continuación tienes un **diagrama conceptual** que podrás recrear directamente. Representa la **jerarquía mínima de un documento HTML5**, con las relaciones padre-hijo entre elementos y los metadatos fundamentales del `<head>`. Después del diagrama, encontrarás una explicación de cada nodo.

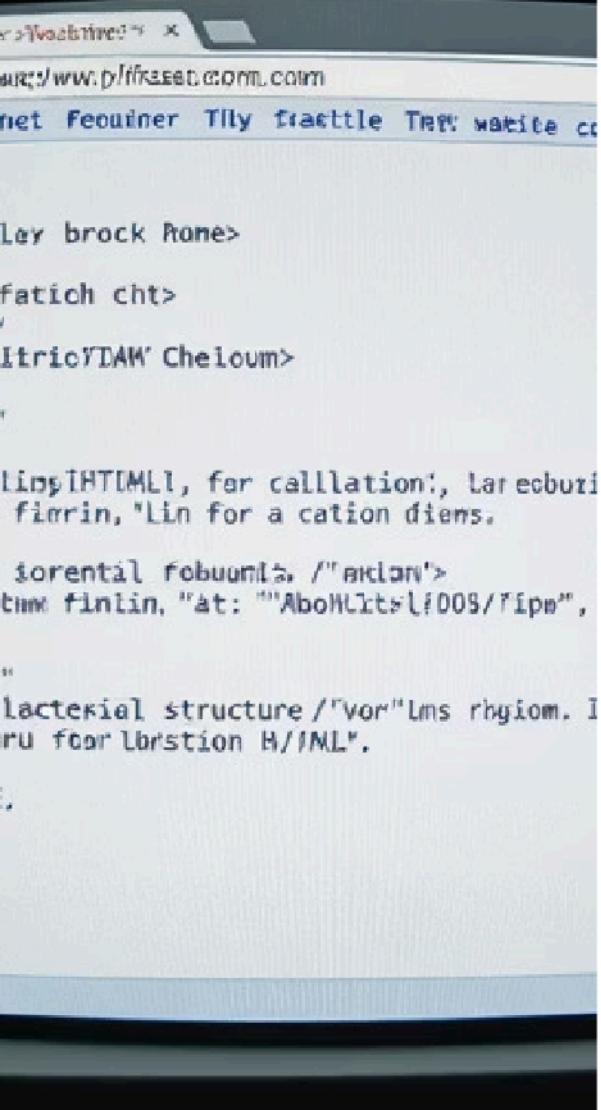


Qué representa cada elemento del esquema:

- `<!DOCTYPE html>`: activa el **modo estándar** del navegador para HTML5. Debe ser lo primero del archivo.
- `<html lang="es">`: elemento raíz; el atributo lang comunica el idioma del contenido a navegadores, lectores de pantalla y buscadores.
- `<head>`: **metadatos** y recursos externos.
- `<meta charset="UTF-8">`: define la codificación y evita problemas con acentos/ñ.
- `<meta name="viewport" ...>`: adapta la página al ancho del dispositivo; imprescindible para **móviles**.
- `<title>`: texto que ves en la **pestaña del navegador** y en los resultados de búsqueda.
- `<link rel="stylesheet" href="styles.css">`: ancla la **hoja de estilos** (opcional en este esquema, pero habitual).
- `<body>`: **contenido visible**.
- `<header>`: cabecera (puede incluir logotipo, nombre del sitio, navegación).
- `<main>`: el **contenido principal único** de la página (solo debe haber uno por documento).
- `<h1>`: título principal del documento (un único `<h1>` recomendado).
- `<p>`: párrafos.
- `<a>`: hipervínculos con href.
- ``: imagen con atributo alt **descriptivo** para accesibilidad.
- `<footer>`: pie de página (autoría, enlaces legales, contacto, etc.).

Sigue esta **jerarquía** en tus archivos para asegurar **compatibilidad, accesibilidad y mantenimiento** a largo plazo.

Caso de Estudio – El primer sitio web de la historia



En 1991, Tim Berners-Lee publicó en el CERN la primera página web del mundo: un documento HTML sobrio y funcional que explicaba qué era la World Wide Web y cómo crear páginas y enlaces. No había CSS, no había imágenes; solo texto y vínculos: exactamente la esencia de HTML. ¿Por qué sigue siendo un caso relevante hoy? Demuestra el propósito original del HTML: estructurar información y conectar documentos mediante hipervínculos.

Pone de manifiesto que la valoración de contenido no depende del adorno visual, sino de la claridad estructural (encabezados, párrafos, listas, enlaces).

Valida que con un mínimo de etiquetas puede existir una web funcional y útil. El estilo y la interactividad llegarían después con CSS y JavaScript, pero la base —HTML— permanece.

Si comparas aquella página con una actual, verás que la columna vertebral es la misma: un <!DOCTYPE>, un que envuelve todo, un con metadatos y un con contenido jerarquizado. Esta continuidad histórica es una pista clara: aprender a dominar lo esencial te prepara para construir sobre ello cualquier proyecto web, del más simple al más complejo.

Herramientas y Consejos Esenciales para un Desarrollo Web Eficaz

Optimiza tu proceso de desarrollo con estas herramientas clave y prácticas recomendadas para trabajar con HTML en VS Code.



Acelera tu flujo con Emmet en VS Code

VS Code trae Emmet preinstalado. Escribe ! y pulsa Tab para generar al instante la plantilla base HTML5 (doctype, html, head, meta charset, meta viewport, title y body). También puedes crear estructuras más complejas:

`ul>li*3` → crea una lista con tres .

`header+main+footer` → esqueleto de bloques principales.



Formatea e indenta sin esfuerzo (Prettier)

Instala Prettier para formatear automáticamente (espacios, saltos de línea, comillas). Un código consistente evita errores de anidación y acelera el trabajo en equipo.



Previsualiza con Live Server

La extensión Live Server (VS Code) levanta un servidor local y recarga en caliente el navegador al guardar. Ideal para iterar rápido mientras construyes la estructura.



Valida con el W3C y linting local

Pasa el documento por W3C Markup Validation Service para detectar etiquetas mal cerradas, atributos inválidos o jerarquías incorrectas. En local, usa HTMLHint para automatizar comprobaciones cada vez que guardas.

Semántica mínima desde el día 1

Aunque esta sesión se centra en lo esencial, acostúmbrate a introducir **semántica básica**: un solo <h1>, jerarquía descendente de encabezados, alt en imágenes, lang en <html>, y si tu documento tiene cabecera/pie, usa <header> y <footer>.

Buenas prácticas de Accesibilidad sin fricción



Idioma del documento

Asegura lang="es" en <html> para que los lectores de pantalla interpreten correctamente el contenido y mejore la segmentación.



Enlaces descriptivos

Utiliza texto de enlace **descriptivo** (evita frases genéricas como "haz clic aquí") para una mejor navegación y comprensión.



Texto alternativo en imágenes

Incluye un alt **significativo** en . Si la imagen es puramente decorativa, usa alt=""". Estas pautas **no cuestan** y elevan la calidad profesional.

Plantilla base recomendada

Un punto de partida breve y correcto que puedes reutilizar para tus proyectos, incluyendo la semántica y accesibilidad básicas:

```
<!DOCTYPE html>
<html lang="es">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Mi Primera Página Web</title>
</head>
<body>
<header>
<h1>Tu nombre o el título del sitio</h1>
</header>
<main>
<p>Breve descripción o bienvenida.</p>
<a href="contacto.html">Ir a contacto</a>

</main>
<footer>
<p>© 2025 Tu Nombre</p>
</footer>
</body>
</html>
```

Mitos y Realidades

X Mito: "HTML es un lenguaje de programación." → **FALSO.**

HTML no ejecuta lógica (no hay condicionales, bucles ni funciones).

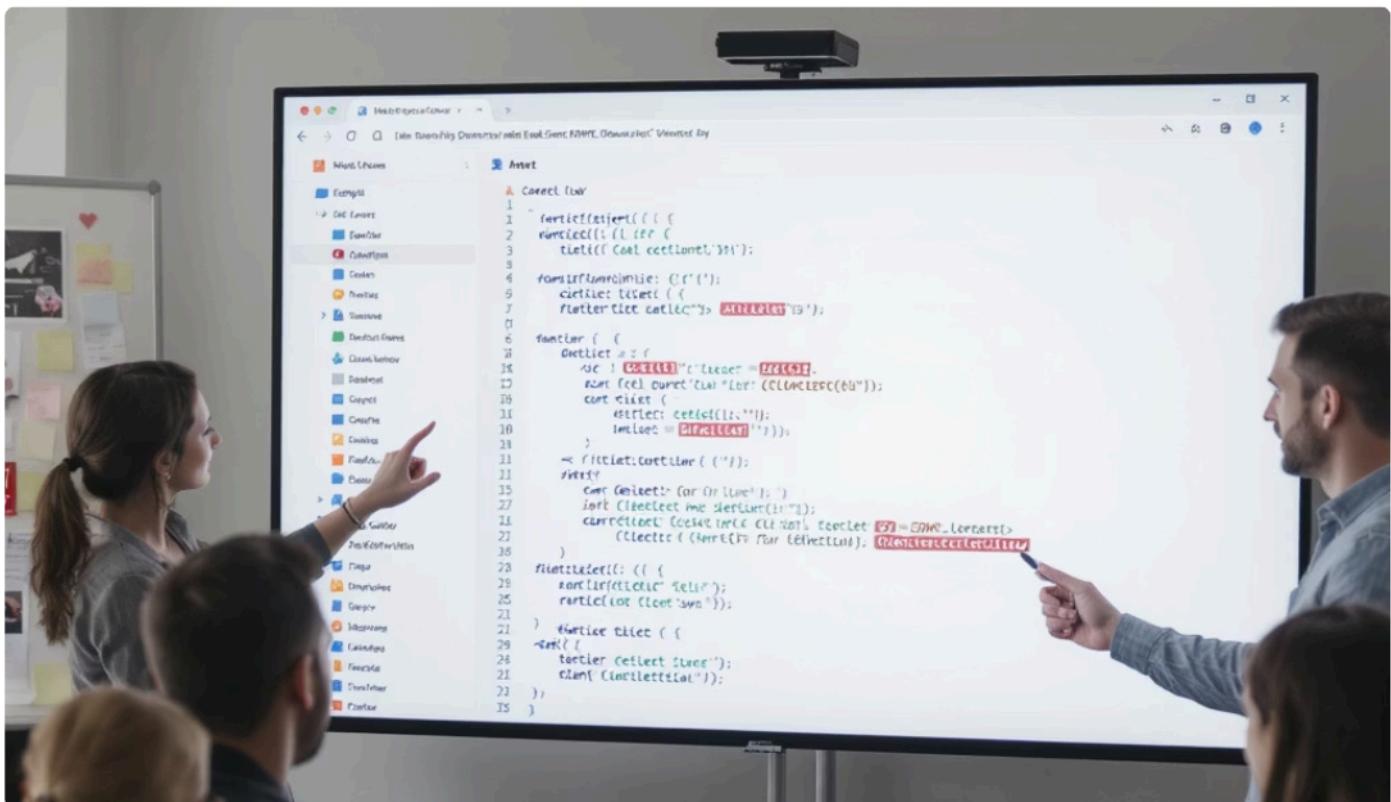
Marca y estructura el contenido. La lógica corre a cargo de JavaScript y los estilos de CSS. Confundirlos lleva a errores de diseño técnico y a expectativas equivocadas sobre lo que puede hacer cada capa.

X Mito: "Da igual la etiqueta si visualmente se ve bien." → **FALSO.**

La semántica importa: mejora la accesibilidad (lectores de pantalla entienden la jerarquía), potencia el SEO (los motores interpretan mejor el contenido) y facilita el mantenimiento (código legible y escalable). No es lo mismo un `<h1>` que un `<p>` con fuente grande: significan cosas distintas para el navegador y para las herramientas de asistencia.

☐ Resumen Final (para examen)

- HTML es lenguaje de marcado: estructura el contenido; no programa.
- Estructura base: `<!DOCTYPE>`, , (metadatos), (contenido).
- Etiquetas esenciales: `h1-h6`, `p`, `a`, `img` con `alt`, listas.
- Usa Emmet, valida con W3C y respeta semántica desde el inicio.



Sesión 4 – Atributos comunes y validación con W3C Validator

En HTML, los **atributos** aportan información adicional a un elemento y se declaran en la **etiqueta de apertura** con la forma `atributo="valor"`. Son claves para tres objetivos profesionales: **estilizar** (CSS), **interactuar** (JavaScript) y **hacer accesible** el contenido (semántica/ARIA). Conocer los **atributos globales** (válidos en casi cualquier elemento) y las **reglas de validación** te permitirá escribir un HTML fiable y fácil de mantener.

Atributos esenciales que debes dominar



id

Identificador **único** en la página. Ideal para apuntar desde CSS/JS o crear enlaces internos (`href="#seccion"`). Tener dos elementos con el mismo id es **error** y rompe accesibilidad (referencias ARIA, etiquetas `<label for="...">`, etc.).



class

Etiqueta **reutilizable** para agrupar y estilizar múltiples elementos. Puedes combinar varias clases en un mismo elemento.



lang (global)

Establece el idioma del contenido; prioritario en `<html>` (`<html lang="es">`) para lectores de pantalla y buscadores.



title (global)

Texto auxiliar, aparece como *tooltip* en muchos navegadores. Úsalo con moderación; no sustituye a un texto visible o a aria-label.



data-*

Atributos personalizados para **transportar datos** que luego leerá JavaScript (por ejemplo, `data-id-usuario="123"`). Son estándares y seguros frente a inventarte atributos no válidos.



Atributos Booleanos

(Sin valor o con el nombre como valor): `required`, `disabled`, `checked`, `readonly`... Se consideran **presentes** con solo declararlos (`<input required>`).

Buenas prácticas de sintaxis

- **Comillas** siempre
(class="boton primario"); evita espacios fuera de las comillas.
- **Un espacio** entre el elemento y el atributo
(``, no `<ahref="...">`).
- **Evita atributos obsoletos**
(align, bgcolor...): su función es de estilo y hoy corresponde a CSS.
- **Respetá la semántica**
Un id único, alt descriptivo en ``, lang correcto y atributos ARIA solo cuando hagan falta.

Validación: por qué y cuándo

El [W3C Markup Validation Service](#) y el [Nu Html Checker](#) (WHATWG) comprueban que tu HTML cumple el estándar. Validar de forma periódica te ayuda a:

1

Detectar errores estructurales

(Etiquetas sin cerrar, id duplicados, atributos inválidos, jerarquías imposibles).

2

Mejorar compatibilidad

Entre navegadores y dispositivos.

3

Asegurar accesibilidad

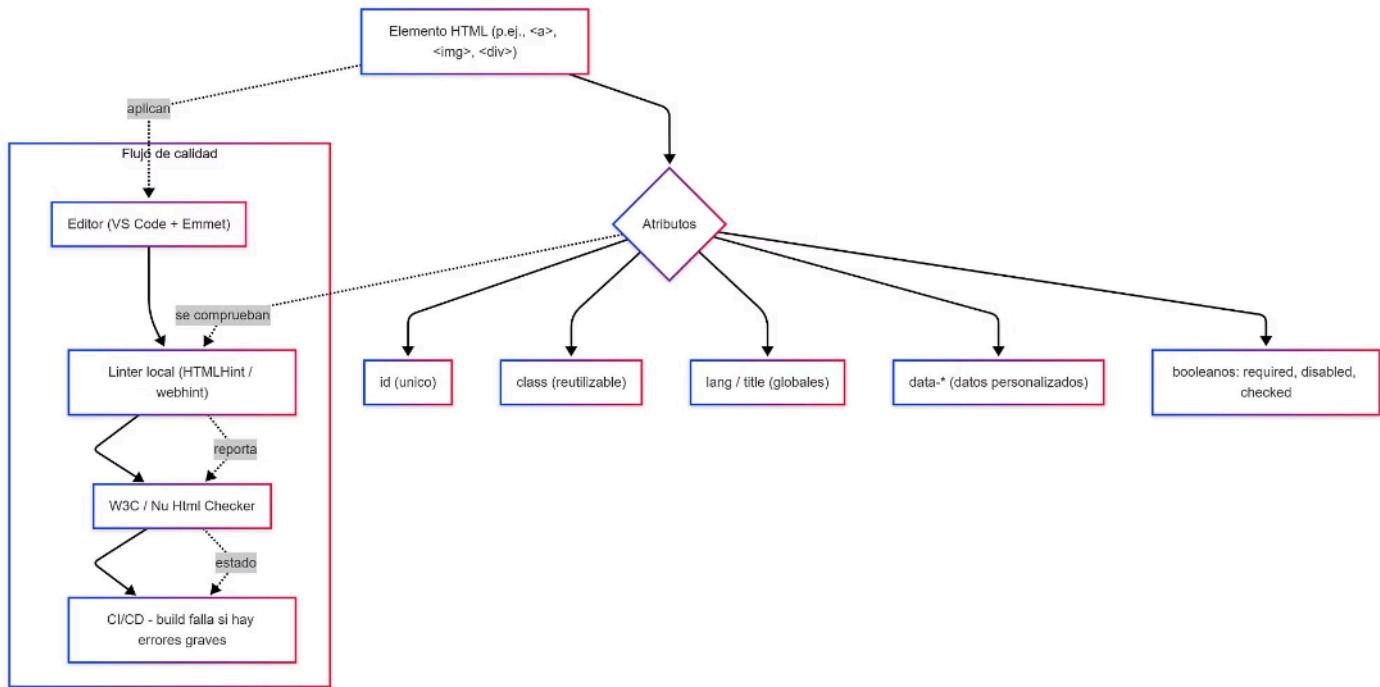
(Relaciones `<label for>`, alt en imágenes, roles ARIA coherentes).

4

Mantener calidad de código

En equipo y preparar despliegues sin sorpresas.

Esquema Visual

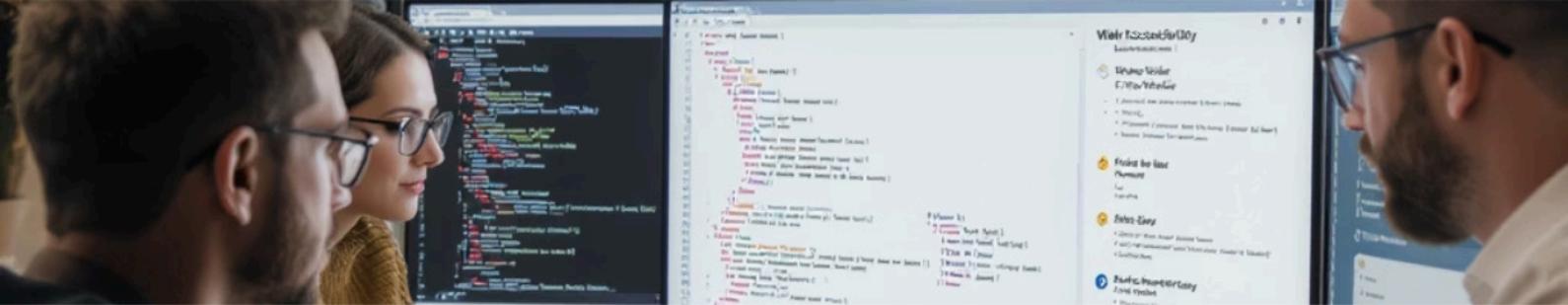


El siguiente diagrama resume la **relación entre elementos y atributos** y el **flujo profesional de validación** (editor → linter → validador → CI). Es suficientemente preciso para recrearlo.

Lectura del diagrama:

A la izquierda, un **elemento HTML** admite distintos **tipos de atributos**: id (único), class (reutilizable), **globales** como lang/title, **custom** data-* y **booleanos**.

A la derecha, el **flujo de calidad** profesional: editas con VS Code, un **linter** local (HTMLHint/webhint) avisa en tiempo real, el **validador** W3C/Nu Html Checker comprueba estándar y, finalmente, la **pipeline de CI** puede **bloquear** el despliegue si hay errores críticos.



Caso de Estudio – Cumplimiento normativo en portales públicos (España)

Contexto

En España, los sitios web del sector público deben cumplir el **Real Decreto 1112/2018** sobre accesibilidad (alineado con **WCAG 2.1** y la norma **UNE-EN 301 549**). Más allá del diseño visual, este cumplimiento exige **HTML correcto y accesible**: lang definido, id únicos, formularios bien etiquetados, alt en imágenes, relaciones ARIA válidas, etc. **Validar el marcado** no es opcional: es parte del **expediente de conformidad** y de los informes de accesibilidad.

Estrategia

Un organismo (pensemos en un portal autonómico con múltiples sedes electrónicas) afrontó una **auditoría de accesibilidad** en varias fases:

1. **Inventario y muestreo:** identificar plantillas y páginas representativas (inicio, trámites, buscador, formularios, micrositios).
2. **Linter local y revisión de plantillas:** incorporar **HTMLHint** y **webhint** en el flujo de desarrollo; ajustar componentes para exigir alt y aria-label cuando proceda, y comprobar **univocidad** de id en componentes repetidos.
3. **Validación continua:** automatizar el **Nu Html Checker** y el **W3C Validator** en CI para romper la build ante errores graves de marcado.
4. **Accesibilidad asistida:** pruebas con **axe DevTools** y **WAVE** para reforzar los aspectos no cubiertos por el validador (contraste, orden de foco, nombres accesibles).
5. **Formación del equipo:** guías de estilo internas con ejemplos de atributos correctos (por ejemplo, usar aria-expanded/aria-controls en acordeones y su sincronización con el estado visible).

Resultado

- **Reducción drástica de errores de validación** en plantillas base (se eliminaron id duplicados y atributos no válidos).
- **Formularios compatibles con lectores de pantalla**, gracias a una aplicación sistemática de <label for="..."> y aria-describedby en mensajes de ayuda.
- **Entrega de informe de conformidad** frente a RD 1112/2018 más ágil al llegar con problemas de marcado **ya resueltos** en CI, acortando plazos de corrección.
- Mejora de **mantenibilidad**: el uso de data-* en lugar de atributos ad-hoc facilitó el trabajo del equipo de JS y evitó advertencias del validador.

Lección clave: la **validación temprana y automatizada** del HTML es un acelerador para el cumplimiento legal y la calidad percibida por la ciudadanía.

Herramientas y Consejos



W3C Markup Validation Service / Nu Html Checker

Valida tu documento por **URL**, **archivo** o **pegar código**. El **Nu Html Checker** (motor moderno del validador) ofrece mensajes detallados y su versión CLI puede integrarse en **CI/CD**. Úsalos para detectar: elementos/atributos obsoletos, anidación inválida, id duplicados, referencias ARIA rotas.



HTMLHint (linter local)

Extensión y CLI para detectar problemas **mientras escribes**: atributos vacíos, comillas ausentes, etiquetas sin cerrar, alt obligatorio en ``. Configura un `.htmlhintrc` con reglas acordadas por el equipo para tener un estándar común.



webhint (by Microsoft)

Analiza **mejores prácticas**: compatibilidad, accesibilidad, PWA, seguridad y también **calidad del HTML**. Funciona como extensión del navegador o CLI. Complementa al validador al dar recomendaciones de **ecosistema** (no solo sintaxis).



VS Code + Emmet + Live Server

- **Emmet**: acelera creación de atributos (`a[href="/contacto"]{Contacto}`) y estructuras repetitivas.
- **Live Server**: recarga automática para iterar rápido y comprobar cambios en atributos (por ejemplo, lang, title, alt, aria-*).



axe DevTools / WAVE

No validan marcado en sentido estricto, pero encuentran **barreras de accesibilidad** relacionadas con atributos: nombres accesibles ausentes, roles incorrectos, aria-* contradictorios o redundantes. Úsalos junto al validador W3C.



Consejos rápidos aplicables ya

- **id vs class**: usa id solo cuando **necesites** un identificador único o un ancla. Para estilos y JS en listas de elementos, **prefiere class**.
- **data-***: guarda datos que tu JS necesita sin inventarte atributos (evitarás errores de validación).
- **Booleanos sin valor**: escribe `<input required>` (no `required="true"`).
- **lang en <html> y hreflang en enlaces alternativos**: ayudan a buscadores y lectores de pantalla.
- **alt en **: describe **función** y **contenido**. Si la imagen es decorativa, `alt=""`.

Cierra siempre tus elementos, mantén **indentación** y evita atributos **obsoletos**: el validador es tu red de seguridad.

Mitos y Realidades

X Mito: "Si en mi navegador se ve bien, el HTML está bien." → **FALSO**

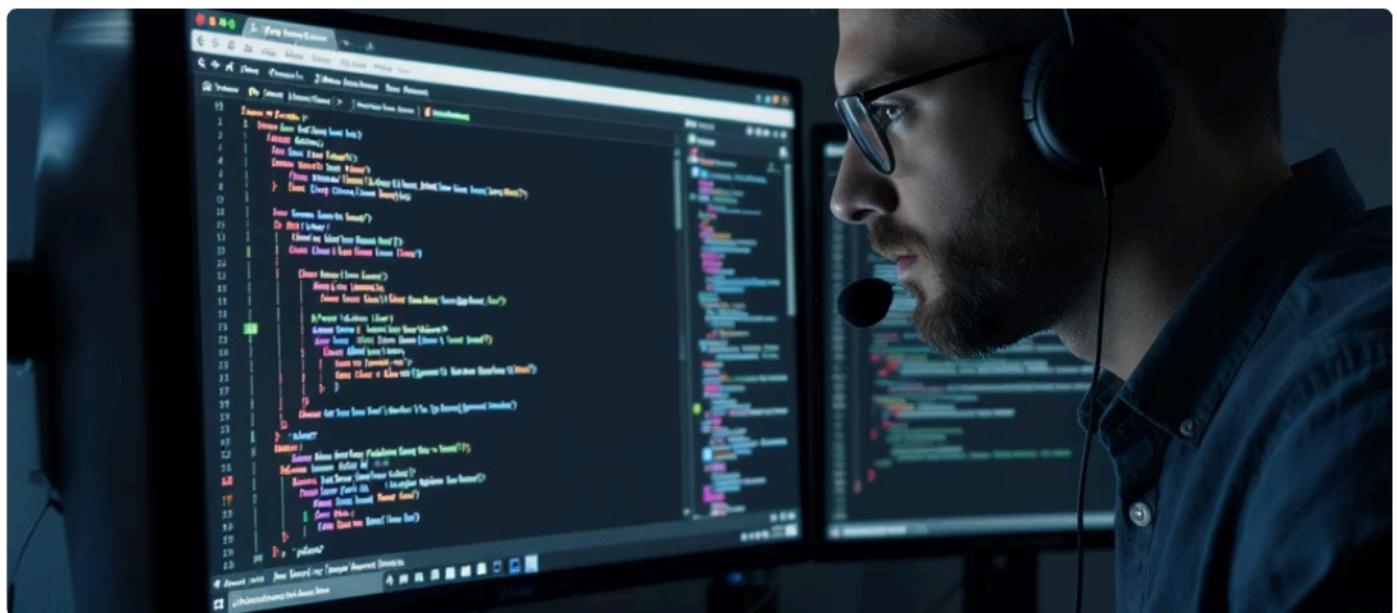
Los navegadores son tolerantes y "arreglan" sobre la marcha muchos errores (etiquetas sin cerrar, anidaciones incorrectas). Eso no significa que el código sea correcto. Puede romperse en otro navegador, afectar a SEO, a la accesibilidad o fallar con scripts que dependen de una estructura válida. Valida y evitarás sorpresas.

X Mito: "La validación W3C es opcional e irrelevante en proyectos reales." → **FALSO**

En entornos profesionales, validar estructura y atributos es crítico: reduce bugs, facilita mantenimiento, mejora compatibilidad y ayuda a cumplir normativa de accesibilidad (obligatoria en el sector público y recomendable en el privado). Integrar el validador en tu CI eleva el umbral de calidad del equipo.

❑ Resumen Final (para examen)

- **Atributos clave:** id (único), class (reutilizable), lang, title, data-***, y booleanos** (required, disabled...).
- **id ≠ class:** usa id solo si necesitas unicidad o anclaje; para estilos/JS en serie, prefiere class.
- **Validación:** usa W3C/Nu Html Checker y apoya con HTMLHint/webhint; ideal integrarlo en CI.
- **Accesibilidad:** lang en <html>, alt correcto, etiquetas semánticas; validación periódica para evitar errores que rompan la experiencia.



Sesión 5 – Etiquetas semánticas modernas (<header>, <article>, <section>, <footer>)

HTML5 marcó un cambio de paradigma al pasar de una web "visual" a una web **semántica**: una web donde las etiquetas **describen su propósito**, no su apariencia. Antes de HTML5, todo se construía con <div> y , sin contexto alguno para navegadores, buscadores ni lectores de pantalla. Hoy, las etiquetas semánticas permiten que los agentes interpretativos (Google, Siri, Alexa, lectores de pantalla, etc.) comprendan **qué representa cada bloque** de contenido.

Qué significa "semántica" en HTML

El término *semántica* viene del griego *semantikos*, "con significado". Aplicado a HTML, implica que **cada etiqueta comunica una función**. Ejemplo:



<header>

Cabecera del sitio o de una sección.



<nav>

Bloque de navegación principal o secundaria.



<main>

Contenido central y único de la página.



<article>

Contenido independiente y autocontenido (noticia, entrada de blog, producto).



<section>

Grupo temático de contenido dentro de un documento.



<aside>

Contenido complementario (barras laterales, widgets, notas).



<footer>

Pie de página o bloque de cierre con autoría, contacto o enlaces.

Diferencias clave

<article>

Representa **una unidad de información completa** que puede distribuirse o entenderse de forma aislada (por ejemplo, una noticia en un periódico digital).

<section>

Agrupa **contenido relacionado** dentro de una página o artículo (por ejemplo, una sección "Deportes" dentro del periódico o "Características" dentro de una ficha de producto).

<header> y <footer> pueden aparecer varias veces: en la página global y dentro de cada <article> o <section>.

Ventajas profesionales

Accesibilidad (A11y)

Los lectores de pantalla reconocen mejor la jerarquía y permiten al usuario saltar entre secciones.

SEO

Los motores de búsqueda comprenden la estructura del documento, jerarquizan encabezados y otorgan mayor relevancia al contenido principal.

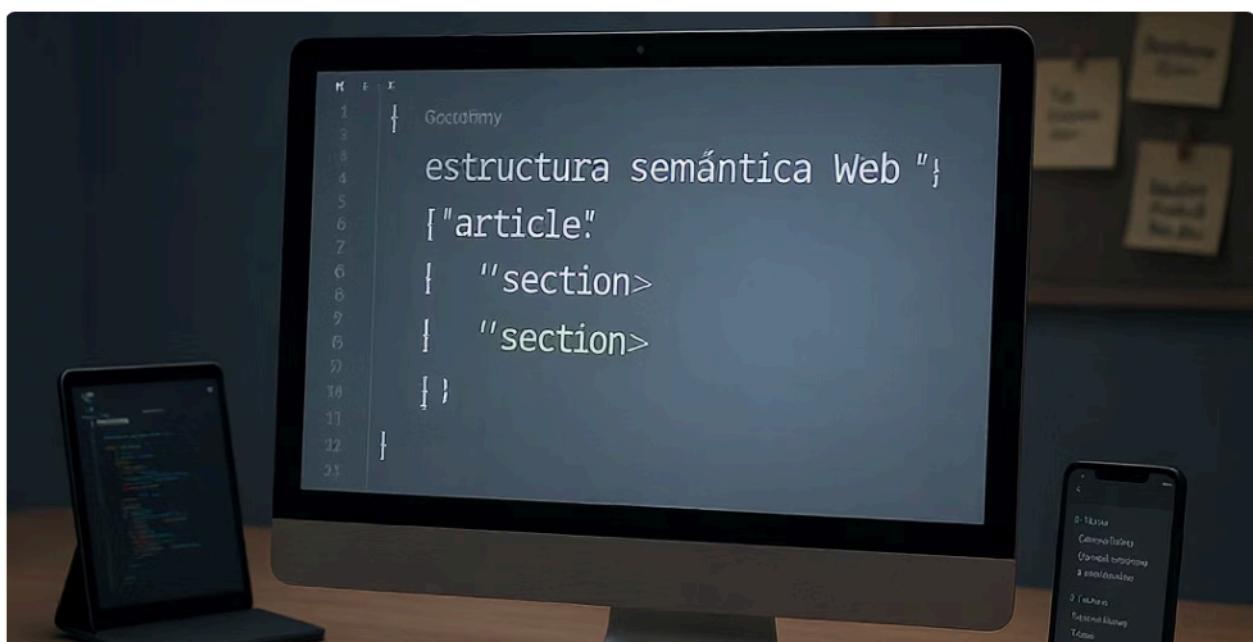
Mantenibilidad

El código es más legible y ordenado; trabajar en equipo es más sencillo.

Interoperabilidad

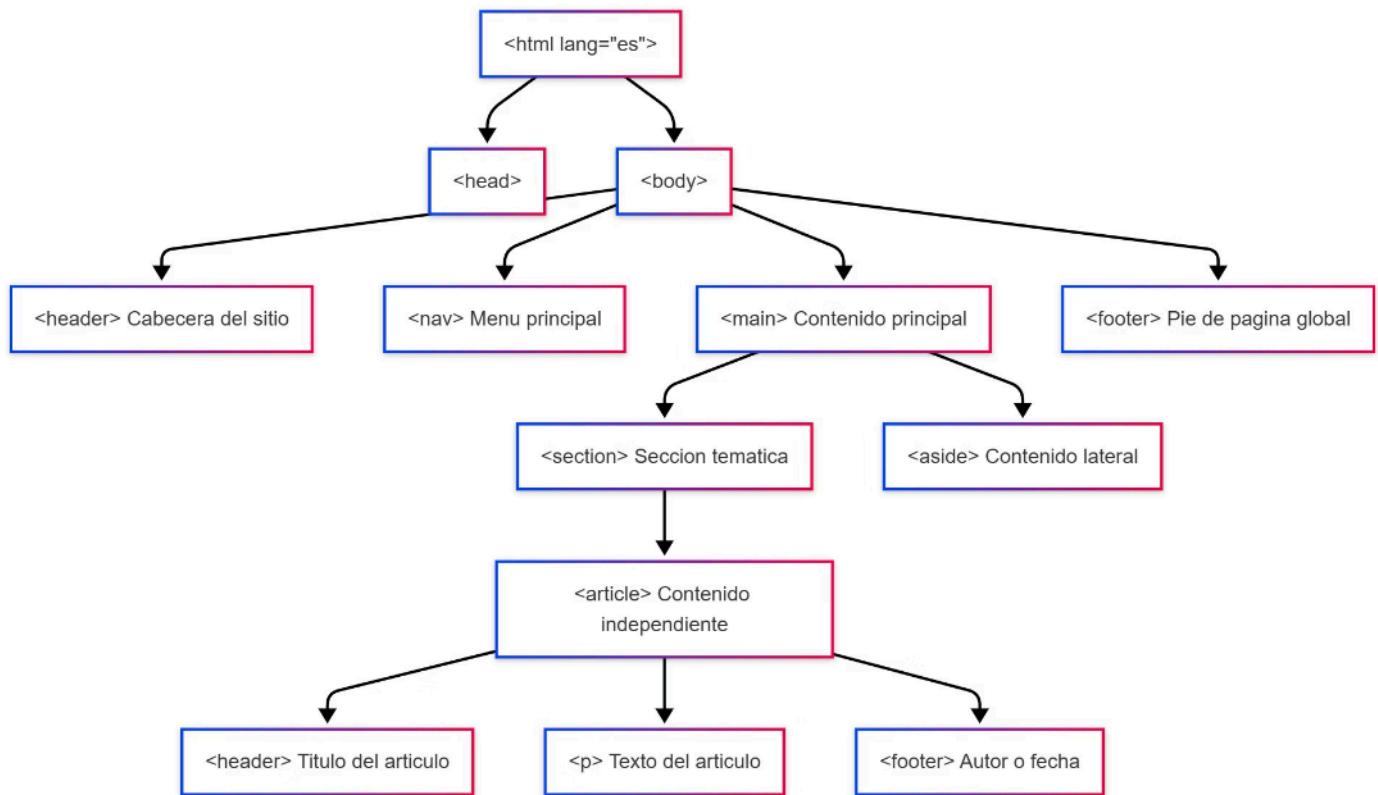
Mejora la indexación en servicios externos (Google News, redes sociales, asistentes de voz).

En definitiva, usar etiquetas semánticas no cambia la apariencia visual por sí sola, pero **mejora radicalmente la calidad técnica** de una web. No se trata de estética, sino de **comunicación estructural** entre tu contenido y el ecosistema digital.



Esquema Visual

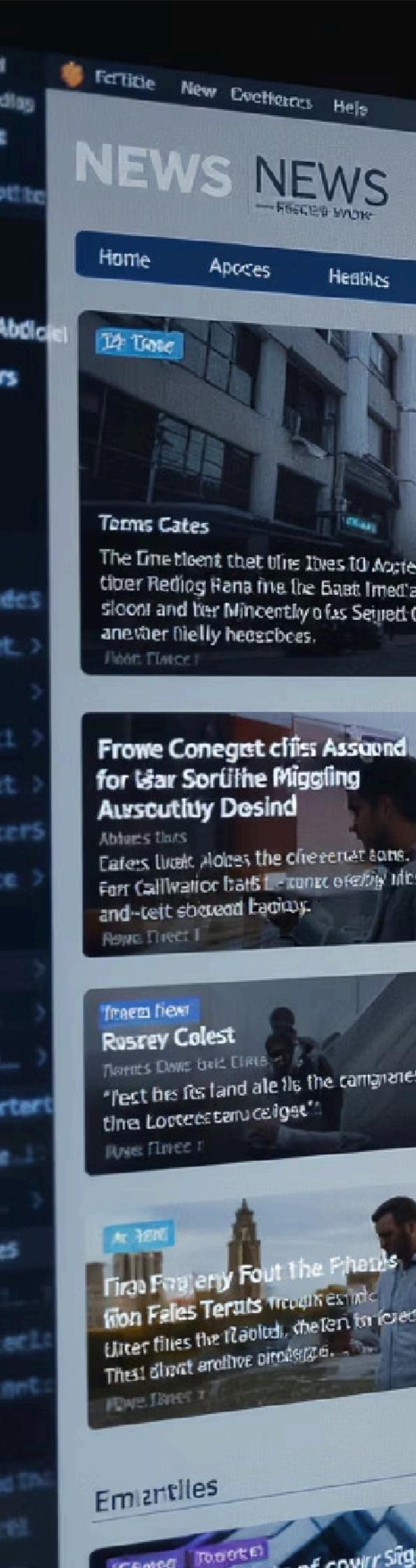
El siguiente diagrama representa la **estructura típica de una página web moderna** utilizando etiquetas semánticas. Permite visualizar el rol jerárquico de cada sección y cómo se interconectan.



Interpretación del esquema:

- <header> (arriba) → cabecera con logotipo, título o barra de navegación principal.
- <nav> → navegación global o interna.
- <main> → único por documento; alberga el contenido relevante.
- <section> → bloque temático (por ejemplo, "Noticias", "Blog", "Productos").
- <article> → unidad de contenido dentro de una sección; puede tener su propio <header> y <footer>.
- <aside> → información complementaria (publicidad, enlaces relacionados, widgets).
- <footer> (global) → cierre del documento con información institucional o contacto.

Este esquema refleja un **layout semántico estándar** que cumple con accesibilidad, SEO y coherencia estructural. El navegador puede interpretar correctamente los límites y propósitos de cada bloque sin necesidad de clases arbitrarias.



Caso de Estudio — Los periódicos digitales (El País y The Guardian)

Contexto

Los portales de noticias fueron pioneros en adoptar las etiquetas semánticas de HTML5, ya que su estructura editorial encaja perfectamente con el modelo <header>, <article>, <section>, <footer>.

El caso de **El País** (elpais.com) es especialmente ilustrativo: cada noticia es un <article>, con su propio <header> que contiene el titular (<h1> o <h2>), una fecha y autor. La página de inicio está formada por múltiples <article> agrupados en <section> (Nacional, Internacional, Cultura, Deportes...).

Estrategia

El equipo técnico de El País reestructuró su plantilla de artículos y portada para:

- Usar <article> para cada noticia, incluso dentro de listados, de modo que los motores de búsqueda puedan indexarlas independientemente.
- Introducir <header> en cada artículo con titular y subtítulo, y <footer> con autor y hora de publicación.
- Agrupar noticias temáticamente dentro de <section> ("Última hora", "Opinión", etc.).
- Añadir <main> para indicar a los lectores de pantalla cuál es el contenido central.
- Mantener un <nav> persistente para la navegación principal.

Resultado

- **SEO:** mejora en la indexación de artículos y mayor presencia en Google News, gracias al reconocimiento de <article> y <header>.
- **Accesibilidad:** los usuarios de lectores de pantalla pueden navegar directamente por artículos, saltar entre secciones o ir al contenido principal con un solo comando.
- **Mantenibilidad:** los equipos de diseño y desarrollo comparten una estructura clara y escalable.
- **UX:** mayor coherencia visual, orden lógico de lectura y navegación más fluida.

Este caso demuestra que la semántica no es decorativa: es una inversión en calidad técnica, visibilidad y usabilidad.

4. Herramientas y Consejos

HTML5 Outliner (extensión de navegador)

Permite visualizar la jerarquía de encabezados y secciones de una página. Ideal para comprobar si tu estructura semántica tiene sentido. Detecta errores comunes como encabezados fuera de orden o ausencia de <main>.

Web Developer Toolbar

Extensión disponible para Chrome y Firefox. En la pestaña *Information* → *View Document Outline* puedes ver la estructura semántica del sitio y analizar si tus <header> y <section> están bien anidados.

Wave y axe DevTools

Herramientas de accesibilidad que muestran si las etiquetas están correctamente interpretadas por los lectores de pantalla y si el orden de lectura es lógico.

Guía práctica: article vs section

- Usa <article> cuando el contenido **tenga sentido por sí mismo**, incluso si lo extraes del contexto (una noticia, post, tarjeta de producto).
- Usa <section> para **organizar bloques temáticos** dentro del documento.
- Puedes tener un <article> dentro de una <section> y viceversa según la jerarquía del contenido.



Estructura mínima recomendada (plantilla base):

```
<body>
<header>
<h1>Mi sitio web</h1>
<nav>
<ul>
<li><a href="#inicio">Inicio</a></li>
<li><a href="#blog">Blog</a></li>
<li><a href="#contacto">Contacto</a></li>
</ul>
</nav>
</header>
<main>
<section id="blog">
<header>
<h2>Últimas publicaciones</h2>
</header>
<article>
<header>
<h3>Cómo mejorar tu código HTML</h3>
</header>
<p>En este artículo analizamos buenas prácticas de HTML5...</p>
<footer>
<p>Por: Ana Torres – 8 oct 2025</p>
</footer>
</article>
</section>
</main>
<footer>
<p>© 2025 Mi sitio web. Todos los derechos reservados.</p>
</footer>
</body>
```

Esta estructura es **completamente válida**, accesible y semántica. Sin CSS ya ofrece un orden lógico perfecto para navegadores, bots y lectores de pantalla.

Evita los errores más comunes:

- Usar `<div>` para encabezados o pies de página cuando debería ser `<header>` o `<footer>`.
- Colocar varios `<main>` por documento (solo uno es correcto).
- Encadenar `<section>` sin encabezado interno. Cada sección debe tener su propio `<h2>` o similar.
- Omitir `<nav>` en favor de un simple `` para menús principales (pierdes contexto semántico).

Mitos y Realidades

X Mito: "Las etiquetas semánticas cambian el diseño de la página."

FALSO. Por defecto, `<header>`, `<section>`, `<article>` o `<footer>` se comportan igual que un `<div>` (bloques de nivel). No añaden estilos visuales ni márgenes. Su valor está en la **estructura y significado**, no en el aspecto. El diseño lo controla CSS, no HTML.

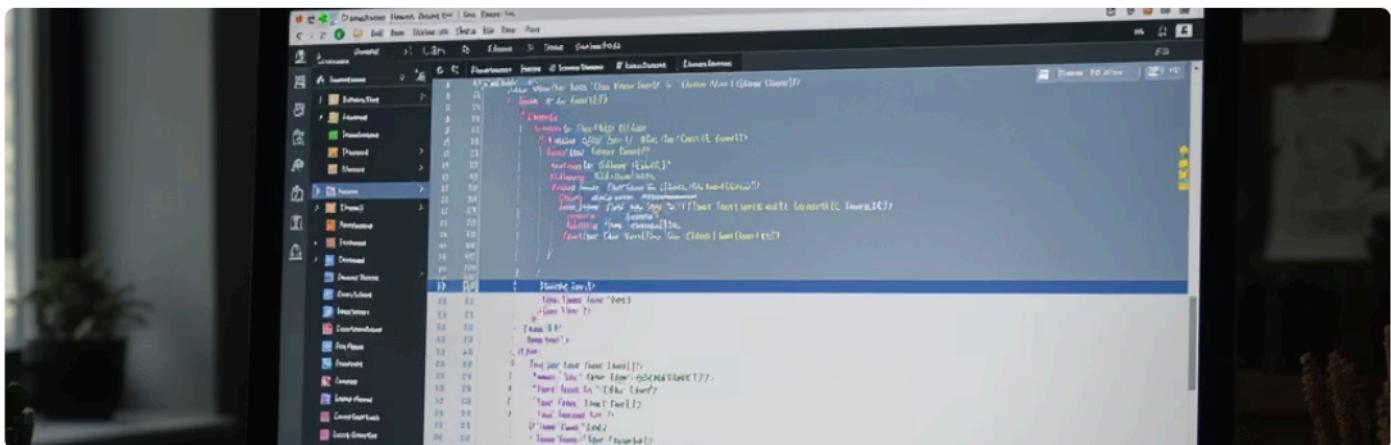
X Mito: "En HTML5 ya no se deben usar `<div>`".

FALSO. Los `<div>` siguen siendo útiles como **contenedores neutrales**, especialmente para agrupar elementos con fines de maquetación o estilos sin significado semántico. La regla es simple:

- Si el bloque tiene **propósito estructural o informativo**, usa una etiqueta semántica.
- Si el bloque es **puramente visual o técnico**, usa `<div>`.

□ Resumen Final (para examen)

- HTML5 introdujo **etiquetas semánticas** para describir el significado del contenido.
- Claves: `<header>` (cabecera), `<footer>` (pie), `<nav>` (navegación), `<main>` (contenido principal), `<article>` (contenido independiente), `<section>` (bloque temático), `<aside>` (complementario).
- Beneficios: **accesibilidad, SEO, legibilidad y mantenimiento profesional**.
- Los `<div>` siguen siendo válidos para maquetación sin valor semántico.
- Caso El País: mejora en SEO y accesibilidad al estructurar noticias con `<article>` y `<section>`.



Sesión 6 – Multimedia en HTML5 (<audio>, <video>, <figure>, <canvas>).

Hasta hace poco, insertar audio o vídeo en una página web requería **plugins externos** como Flash o QuickTime, que eran lentos, inseguros y no funcionaban en dispositivos móviles. Con la llegada de **HTML5**, el contenido multimedia se integró de forma **nativa en el navegador**, sin depender de complementos. Esto transformó la forma en que consumimos información en la web: hoy puedes ver vídeos, escuchar podcasts o reproducir animaciones directamente en el navegador.

HTML5 incorpora cuatro etiquetas clave para la gestión multimedia:

- <video>: reproduce archivos de vídeo directamente en la página.
- <audio>: permite incluir archivos de sonido o música.
- <figure>: agrupa un recurso visual o multimedia junto con su descripción.
- <canvas>: proporciona un *lienzo* en blanco donde se pueden dibujar gráficos, animaciones o visualizaciones con JavaScript.

La etiqueta <video>

Permite insertar vídeos en formato MP4, WebM u Ogg.

Su estructura básica es:

```
<video controls width="640" height="360">
<source src="video.mp4" type="video/mp4">
<source src="video.webm" type="video/webm">
Tu navegador no soporta el vídeo HTML5.
</video>
```

- `controls` muestra los controles de reproducción (play, pausa, volumen...).
- `poster` permite mostrar una **imagen de portada** antes de reproducir el vídeo.
- `autoplay`, `loop` y `muted` gestionan comportamiento automático (aunque se deben usar con cuidado por accesibilidad).

La etiqueta <audio>

Funciona igual que <video>, pero sin imagen. Ejemplo:

```
<audio controls>
<source src="podcast.mp3" type="audio/mpeg">
<source src="podcast.ogg" type="audio/ogg">
Tu navegador no soporta el audio HTML5.
</audio>
```

Admite formatos como MP3, WAV y Ogg. Es ideal para **podcasts, efectos sonoros o música de fondo**.

La etiqueta <figure>

Sirve para **empaquetar** cualquier contenido visual (imagen, vídeo, gráfico) con su **leyenda explicativa** dentro de <figcaption>.

Ejemplo:

```
<figure>

<figcaption>Paisaje costero capturado al anochecer.</figcaption>
</figure>
```

Usar <figure> mejora la **accesibilidad y el SEO**, ya que vincula semánticamente el recurso con su descripción.

La etiqueta <canvas>

Define una zona rectangular donde puedes **dibujar dinámicamente** mediante JavaScript: gráficos, animaciones, juegos, visualizaciones de datos, etc.

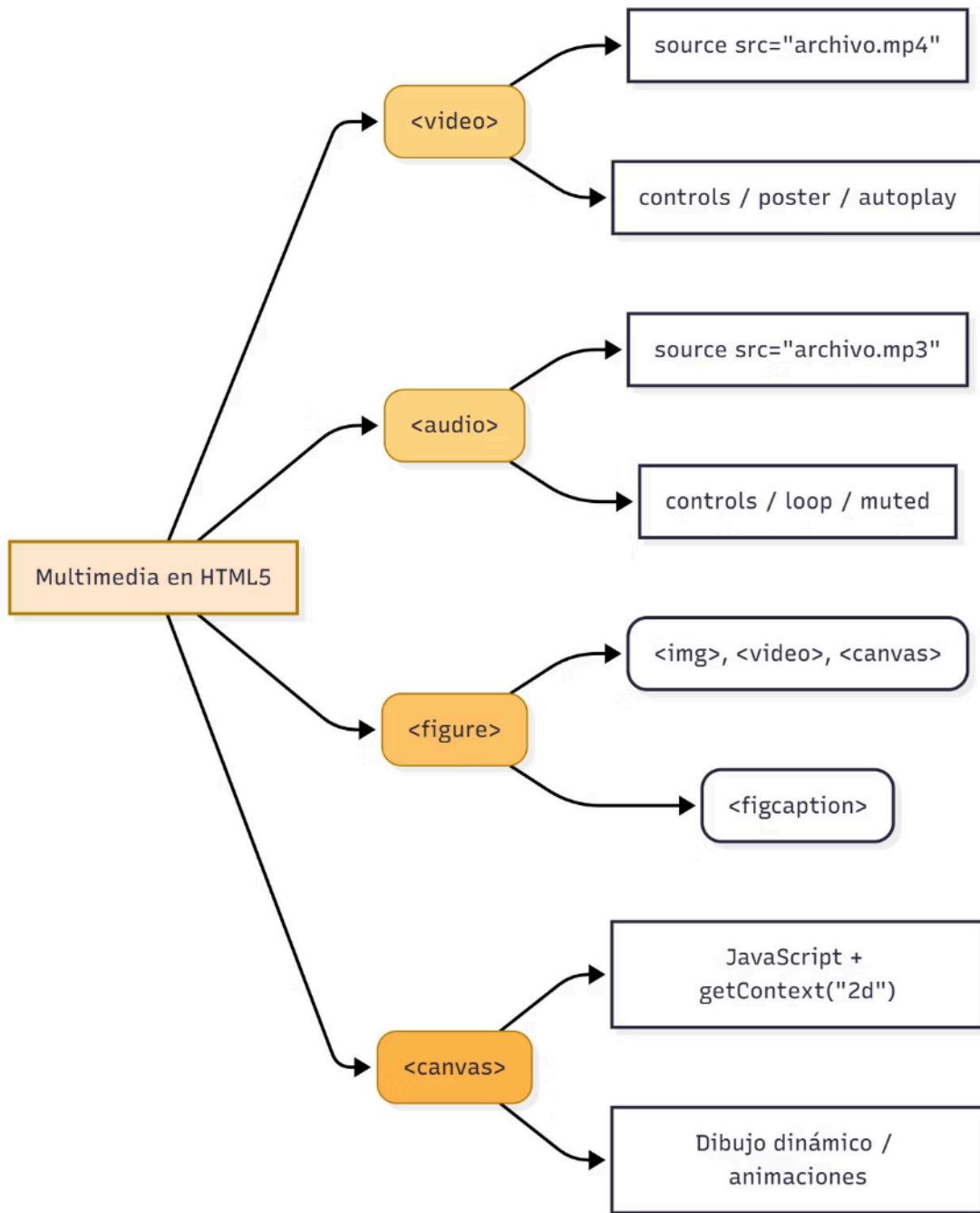
```
<canvas id="miCanvas" width="400" height="300"></canvas>
<script>
const c = document.getElementById("miCanvas");
const ctx = c.getContext("2d");
ctx.fillStyle = "blue";
ctx.fillRect(50, 50, 200, 100);
</script>
```

- getContext("2d") crea un entorno de dibujo bidimensional.
- Los contenidos de <canvas> **no son accesibles ni indexables** por motores de búsqueda: es solo un contenedor gráfico.

En resumen, HTML5 transformó la experiencia digital al **eliminar barreras tecnológicas**. Hoy, cualquier página moderna puede ofrecer vídeo, audio y animaciones sin depender de terceros, con mayor **rendimiento, seguridad y accesibilidad**.

Esquema Visual

El siguiente diagrama muestra las relaciones entre las etiquetas multimedia en HTML5, su función y los atributos más relevantes.



Interpretación:

- **<video>** y **<audio>** comparten estructura y atributos similares.
- **<figure>** actúa como contenedor semántico para elementos visuales, con **<figcaption>** como texto asociado.
- **<canvas>** permite crear contenido gráfico interactivo mediante JavaScript.

Todo el bloque pertenece al concepto de "Multimedia HTML5", que ofrece reproducción nativa, semántica y accesible.



Caso de Estudio – YouTube y la transición de Flash a HTML5

Contexto

Durante más de una década, **YouTube** usó el reproductor Flash de Adobe para mostrar videos. Pero este sistema era problemático:

- No funcionaba en dispositivos móviles (Apple nunca permitió Flash en iOS).
- Requería actualizaciones constantes y generaba **vulnerabilidades de seguridad**.
- Consumía más recursos y ralentizaba los navegadores.

Estrategia

En 2015, Google anunció la **migración completa a HTML5** como reproductor por defecto. Las razones principales:

1. **Compatibilidad universal:** los videos HTML5 funcionan en navegadores modernos, móviles y Smart TVs sin plugins.
2. **Mejor rendimiento:** el uso de códecs optimizados (H.264, VP9) redujo el consumo de CPU.
3. **Integración con nuevas APIs:** como *Media Source Extensions (MSE)* y *Encrypted Media Extensions (EME)*, que permiten streaming adaptativo y contenido protegido (DRM).
4. **Accesibilidad:** soporte nativo para subtítulos (<track kind="subtitles">), descripción de audio y controles visibles por teclado.

Resultado

- **Reducción del 30 %** en fallos de reproducción y tiempos de carga más rápidos.
- **Compatibilidad total con dispositivos móviles**, que representaban ya más del 50 % del tráfico.
- **Mayor seguridad** al eliminar dependencias externas.
- **Estandarización global:** el <video> HTML5 se consolidó como estándar de la industria.

La transición de YouTube marcó un antes y un después: hoy, todos los grandes servicios de vídeo — Netflix, Twitch, Vimeo, Disney+ — utilizan HTML5 como base tecnológica.

Herramientas y Consejos

Conversores de formato

No todos los navegadores soportan los mismos códigos. Para garantizar la compatibilidad, convierte tu vídeo a varios formatos.

-  **HandBrake** (gratuito, multiplataforma): convierte entre MP4, WebM y Ogg.
-  **FFmpeg** (línea de comandos): profesional y automatizable para lotes de vídeos.
-  **CloudConvert**: servicio web rápido si no quieres instalar nada.

Atributos y buenas prácticas

- **controls**: siempre visible por accesibilidad (no ocultes controles sin ofrecer alternativa).
- **poster="imagen.jpg"**: mejora la experiencia visual mientras el vídeo carga.
- **preload**: puede ser none, metadata o auto (controla la carga inicial).
- **muted autoplay**: los navegadores solo permiten autoplay si el vídeo está silenciado por defecto.
- **loop**: útil para clips breves o vídeos decorativos.

Subtítulos y accesibilidad

Usa la etiqueta para incluir subtítulos o descripciones:

```
<video controls>
<source src="video.mp4"
        type="video/mp4">
<track kind="subtitles"
      src="subtitulos.vtt"
      srclang="es"
      label="Español">
</video>
```

Los archivos .vtt (WebVTT) permiten sincronizar texto con el vídeo y mejoran la accesibilidad.



Canvas y visualizaciones interactivas

El canvas es ideal para dashboards, juegos, simulaciones o animaciones ligeras. Frameworks recomendados: p5.js, Chart.js, Fabric.js.

- Usa `requestAnimationFrame()` en lugar de `setInterval()` para animaciones más suaves.
- Recuerda: no uses canvas para contenido esencial; los gráficos deben tener una alternativa textual.



Validación y compatibilidad

- Comprueba formatos y atributos válidos en MDN Web Docs – Video and Audio.
- Valida tu HTML con el W3C Validator: detectará si usas atributos obsoletos o anidados incorrectamente.
- Usa Can I Use (caniuse.com) para verificar la compatibilidad de códigos y APIs multimedia con los distintos navegadores.



Optimización de rendimiento

- Comprime los archivos multimedia antes de subirlos.
- Añade el atributo `loading="lazy"` en imágenes dentro de video para mejorar el tiempo de carga.
- Usa CDN (Content Delivery Network) para distribuir contenido pesado (Cloudflare, AWS CloudFront).

5. Mitos y Realidades

X Mito: "Con <video> puedo reproducir cualquier formato." → **FALSO.** Cada navegador admite códigos distintos: Chrome y Firefox prefieren WebM/VP9, mientras que Safari usa MP4/H.264. Siempre ofrece **múltiples fuentes** dentro del <video> para asegurar compatibilidad.

X Mito: "<canvas> funciona igual que ." → **FALSO.** <canvas> es un **lienzo vacío** controlado por JavaScript. No tiene contenido visual propio ni texto alternativo. Los gráficos generados no son accesibles ni indexables. muestra un recurso estático con significado; <canvas> muestra una **representación dinámica**.

❑ Resumen Final (para examen)

- **Etiquetas multimedia clave:** <video>, <audio>, <figure>, <figcaption>, <canvas>.
- **Ventajas:** reproducción nativa, sin plugins, compatible y accesible.
- **Caso real:** YouTube sustituyó Flash por HTML5 → mejor rendimiento, compatibilidad y seguridad.
- **Buenas prácticas:** ofrecer varios formatos (<source>), incluir subtítulos (<track>), usar <figure> con descripción, y optimizar peso y accesibilidad.
- **Recuerda:** <canvas> requiere JavaScript y no debe usarse para información textual o esencial.



Sesión 7 – Formularios modernos, *inputs* HTML5 y validación nativa

Los formularios son la **puerta de entrada de datos** en la web: permiten registrarse, iniciar sesión, enviar comentarios, realizar compras o interactuar con servicios en línea. HTML5 los revolucionó incorporando **nuevos tipos de campos (*input types*)**, validación automática sin JavaScript y controles nativos pensados para **dispositivos móviles y accesibilidad**.

Un formulario es una combinación de elementos:

```
<form action="procesar.php" method="post">
<label for="nombre">Nombre:</label>
<input type="text" id="nombre" name="nombre" required>
<label for="correo">Correo:</label>
<input type="email" id="correo" name="correo" required>
<label for="fecha">Fecha de nacimiento:</label>
<input type="date" id="fecha" name="fecha">
<button type="submit">Enviar</button>
</form>
```

Claves del ejemplo:

- <form> es el contenedor y define **adónde se envían los datos** mediante los atributos action (URL) y method (GET o POST).
- <label> describe el propósito del campo y mejora la accesibilidad.
- <input> crea los campos de texto o selección.
- required obliga al usuario a completarlo antes de enviar.
- <button> ejecuta la acción de envío.

Nuevos tipos de *input* en HTML5

HTML5 añadió tipos diseñados para **mejorar la experiencia de usuario (UX)**, especialmente en móviles.

email	<input type="email">	Comprueba automáticamente el formato del correo.
url	<input type="url">	Verifica que el texto sea una dirección web.
tel	<input type="tel">	Muestra teclado numérico en móviles.
number	<input type="number">	Permite subir/bajar valores con flechas.
date, time, datetime-local	<input type="date">	Selector de calendario o reloj.
range	<input type="range" min="0" max="10">	Control deslizante para valores numéricos.
color	<input type="color">	Selector de color nativo.

Validación nativa en HTML5

Antes de HTML5, validar datos requería código JS. Hoy, el navegador puede comprobar automáticamente los valores de los campos gracias a los **atributos de validación nativos**, entre ellos:

Atributo	Función
required	Hace obligatorio un campo.
pattern	Expresión regular para definir el formato (ej. pattern="[0-9]{9}" para un DNI).
min / max	Límite mínimo o máximo para números o fechas.
minlength / maxlength	Longitud mínima o máxima del texto.
step	Intervalo de incremento para number o range.

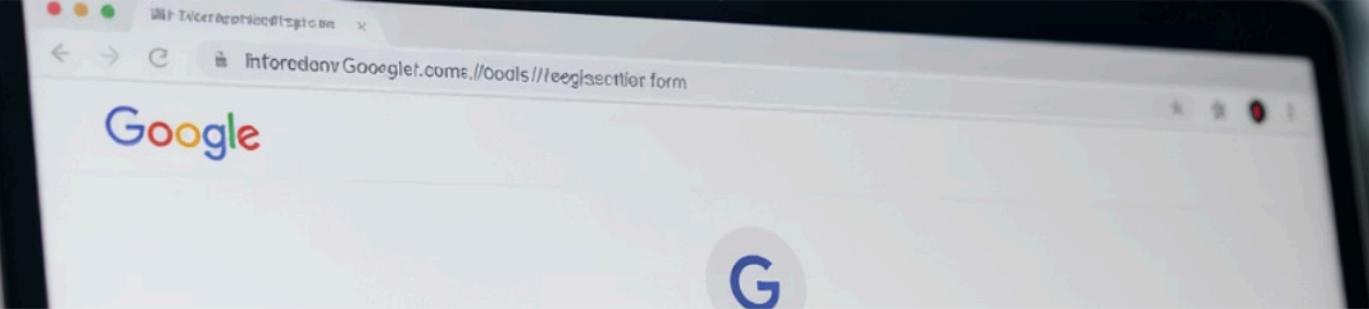
El navegador muestra mensajes automáticos de error cuando se incumplen las condiciones, sin necesidad de escribir una sola línea de JS.

Accesibilidad y semántica

Un formulario accesible usa tres pilares:

1. **Asociar etiquetas (`<label>`)** con los campos mediante el atributo for que coincide con el id.
2. **Evitar depender solo de color o posición visual**: los campos deben estar etiquetados.
3. **Mensajes claros**: el texto de error debe explicar qué falta o está mal, no solo marcarlo en rojo.

Una buena estructura de formulario no solo funciona: **se comunica claramente con todos los usuarios**.



Caso de estudio – Los formularios de registro de Google

Contexto

El proceso de creación de cuentas de Google es uno de los ejemplos más refinados de formularios web: rápido, intuitivo y seguro.

Estrategia

Google combina las características de HTML5 con su propio sistema de diseño (Material Design) para garantizar usabilidad y accesibilidad:

1. Uso de tipos nativos:

- type="email" para validar formato de correo automáticamente.
- type="password" con minlength="8".
- type="date" para seleccionar la fecha de nacimiento.

2. Validación nativa + validación en servidor:

- HTML5 muestra mensajes inmediatos si el usuario deja un campo vacío o introduce un formato incorrecto.
- Luego, el servidor comprueba la existencia o unicidad del correo (seguridad).

3. Accesibilidad:

- Cada campo tiene un <label> visible y un atributo aria-describedby para describir errores o consejos.
- Los mensajes de error son **contextuales**, no genéricos.

4. Retroalimentación visual:

- Colores y animaciones suaves informan al usuario sobre los estados: verde = válido, rojo = error, gris = pendiente.

Resultado

- Reducción de un 25 % en los abandonos de formulario.
- Menor carga de soporte técnico gracias a mensajes automáticos de validación.
- Cumplimiento de accesibilidad (WCAG 2.1 nivel AA).
- Fluidez total en móviles: los teclados se adaptan automáticamente al tipo de dato.

La clave del éxito de Google está en **combinar la validación automática del navegador con controles de seguridad en el servidor**, garantizando experiencia y protección.

Herramientas y consejos

Editores y extensiones útiles

- **VS Code + Emmet:** atajos para crear estructuras completas: form>label+input*3+b utton genera al instante un formulario básico.
- **Live Server:** visualiza cambios en tiempo real sin recargar manualmente.
- **HTMLHint:** detecta errores como label sin for o id duplicados.

Herramientas de validación

- **W3C Markup Validation Service:** verifica que tus formularios cumplan con los estándares.
- **axe DevTools / WAVE:** analizan accesibilidad: revisan relaciones entre label y input, roles, y mensajes de error comprensibles.

Atributos imprescindibles

- **required:** hace obligatorio un campo.
- **pattern:** define formatos específicos. Ejemplo:
- **placeholder:** muestra un texto orientativo, pero no reemplaza al label.
- **aria-describedby:** conecta el campo con un mensaje de ayuda o error visible para lectores de pantalla.

CSS y Pseudoclases de Validación

Puedes personalizar visualmente la validación del navegador usando CSS:

```
input:valid { border-color: green; }
input:invalid { border-color: red; }
```

Estas pseudoclases (:valid y :invalid) permiten mostrar al usuario, sin JS, qué campos están correctos o incorrectos.

Buenas Prácticas Profesionales

- No relias solo en el color para comunicar errores. Usa íconos o texto adicional.
- Agrupa campos relacionados dentro de <fieldset> y usa <legend> para describirlos.
- Mantén un orden lógico de tabulación (tabindex) y usa autocomplete para acelerar el llenado.
- Verifica los datos **también en el servidor:** la validación del cliente es una ayuda, no una barrera.

Mitos y realidades

X Mito: "La validación HTML5 es suficiente, no hace falta validación en el servidor."

FALSO. Los atributos como required o pattern mejoran la experiencia del usuario, pero pueden desactivarse fácilmente manipulando el HTML. La validación del servidor siempre es necesaria para garantizar la integridad y seguridad de los datos.

X Mito: "Puedo usar type="text" para todo."

FALSO. Usar el tipo correcto (email, tel, date, number) mejora la accesibilidad y la experiencia en móviles (teclados adaptados). Además, permite al navegador aplicar validaciones automáticas y mostrar la interfaz nativa correspondiente.

□ Resumen final

- **Formularios HTML5** permiten validar sin JavaScript con atributos como required, pattern o minlength.
- Nuevos tipos de *input* (email, tel, date, number, color) mejoran la usabilidad y la accesibilidad.
- Usa siempre <label> asociado al campo (for = id).
- La validación del navegador **no sustituye** la validación del servidor.
- Personaliza los estados con CSS (:valid, :invalid) y usa herramientas como WAVE para asegurar accesibilidad.



Sesión 8 – Hipervínculos, URLs, accesibilidad y SEO básico

Los **hipervínculos** son la esencia de la World Wide Web: permiten **conectar documentos, recursos y sitios** entre sí. Sin ellos, la web sería un conjunto aislado de páginas. En HTML, los enlaces se crean con la etiqueta **<a> (anchor)**, que literalmente actúa como un "ancla" que une un punto con otro.

La estructura básica de un enlace es:

```
<a href="https://www.wikipedia.org">Visita Wikipedia</a>
```

El atributo href (*hypertext reference*) indica **la dirección de destino** del enlace. El texto entre <a> y es lo que el usuario ve y pulsa.

Tipos de URL: absolutas y relativas

URL absoluta: contiene la dirección completa (protocolo, dominio y ruta). Ejemplo: Google

Se usa para enlazar **fuerza de tu propio sitio web**.

URL relativa: apunta a una página o archivo **dentro del mismo sitio**. Ejemplo: Contacto

- Se usa para mantener los enlaces funcionales aunque se cambie el dominio del proyecto.

HTML también permite **enlaces internos** dentro de una misma página usando **anclas**:

```
<a href="#inicio">Ir al principio</a>
...
<h1 id="inicio">Encabezado principal</h1>
```

En este caso, el enlace lleva al elemento que tenga el atributo id="inicio".

Atributos importantes de <a>

- **target="_blank"**: abre el enlace en una nueva pestaña o ventana. Se recomienda acompañarlo de **rel="noopener noreferrer"** para evitar riesgos de seguridad.
- **title**: muestra una descripción breve al pasar el ratón por encima.
- **download**: indica que el enlace descarga un archivo en lugar de abrirlo. Ejemplo: Descargar informe

Accesibilidad y texto descriptivo

Uno de los errores más comunes es usar textos genéricos como "haz clic aquí" o "más información".

El texto del enlace debe **describir el destino o la acción**, porque:

- Los **lectores de pantalla** leen solo el texto del enlace, no su contexto.
- Los motores de búsqueda usan el texto del enlace (*anchor text*) para entender la relación entre páginas.

Ejemplo correcto:

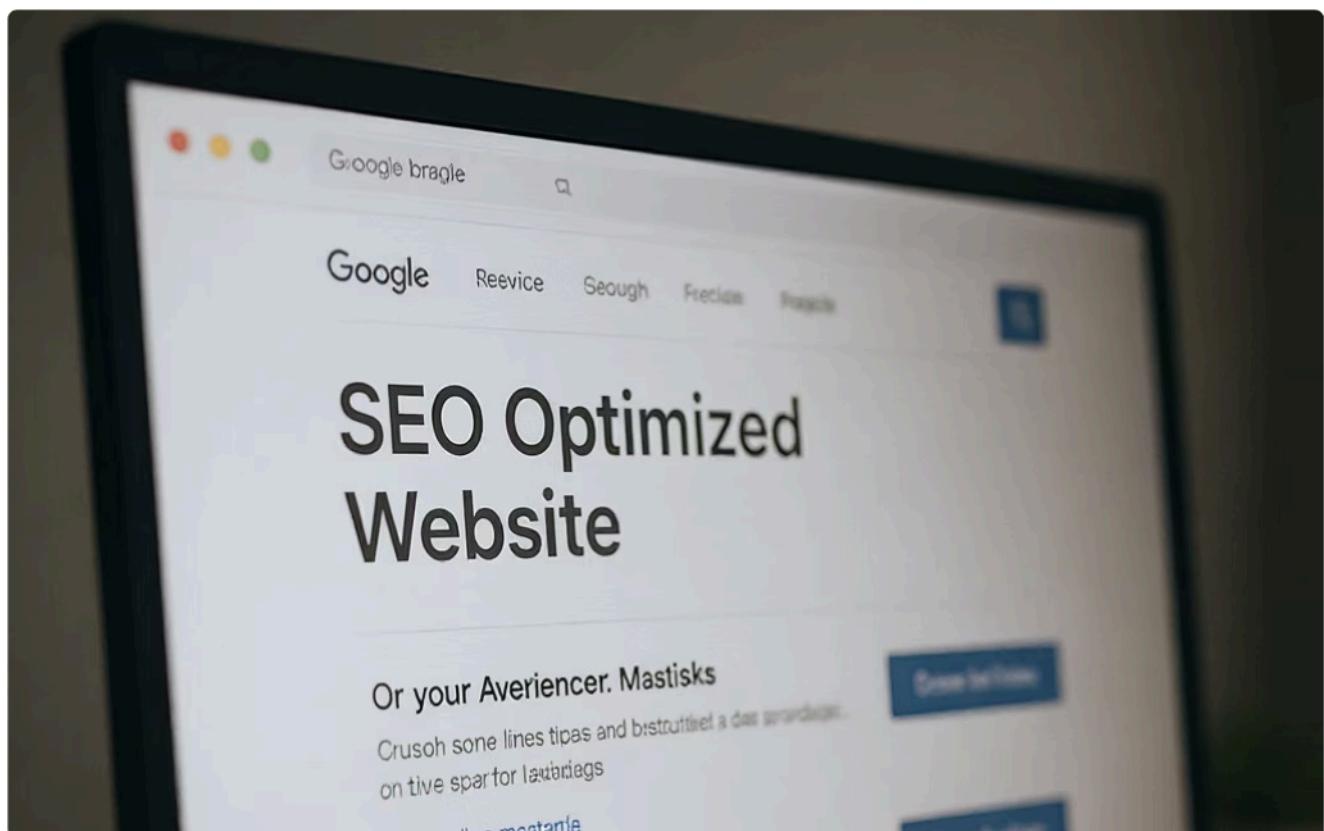
```
<a href="/politica-privacidad.html">Consulta nuestra política de privacidad</a>
```

Enlaces y SEO

Desde el punto de vista del **SEO (Search Engine Optimization)**, los enlaces cumplen dos funciones esenciales:

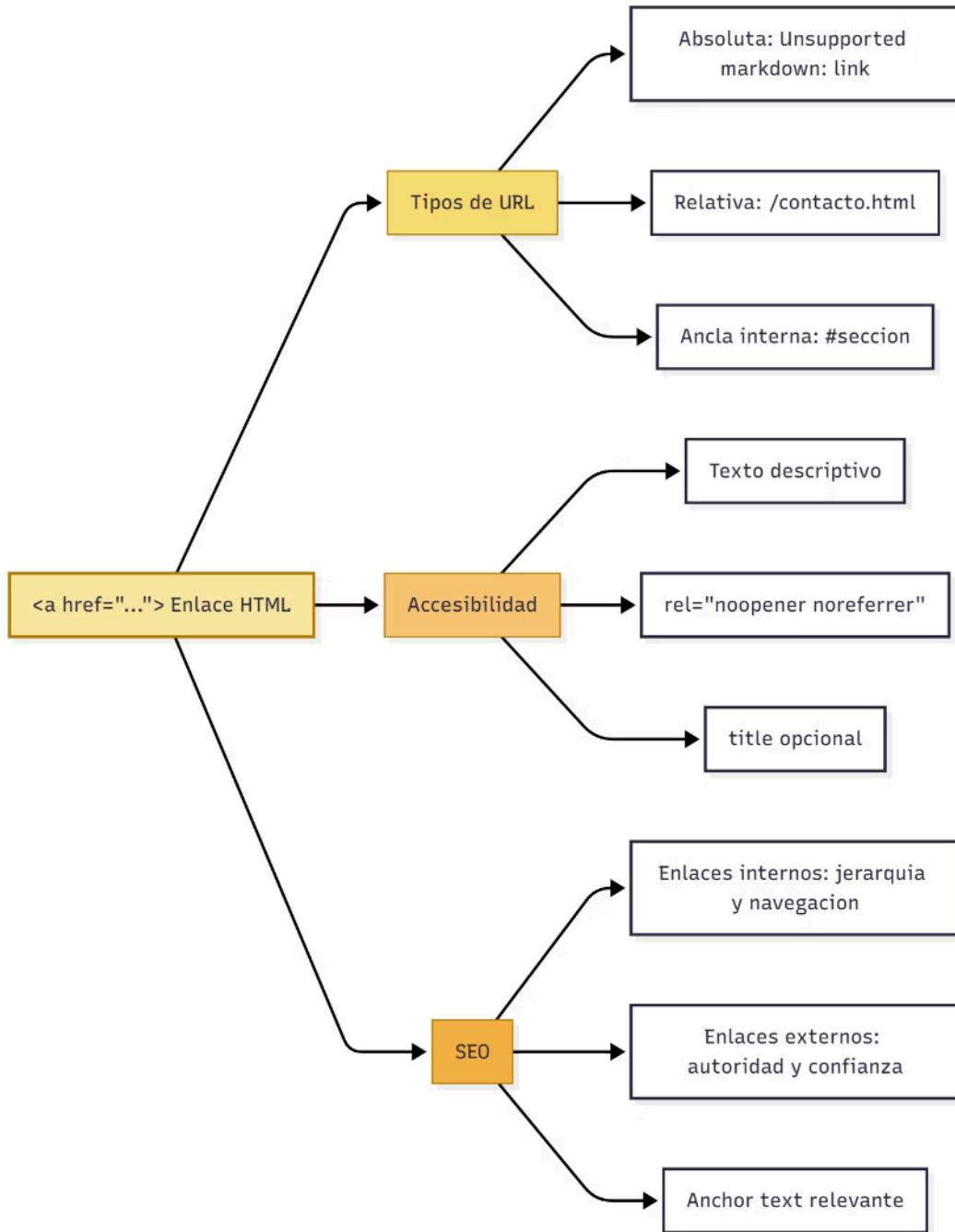
1. **Enlaces internos:** ayudan a los motores de búsqueda a descubrir, indexar y jerarquizar las páginas dentro de tu sitio.
2. **Enlaces externos (backlinks):** aumentan la autoridad de tu dominio si provienen de fuentes fiables y relevantes.

El uso correcto de hipervínculos es un **factor estructural y semántico** que influye directamente en la visibilidad de tu web.



Esquema visual

El siguiente diagrama muestra cómo se relacionan los enlaces, las URLs y los aspectos de accesibilidad y SEO.



Interpretación:

- Todo parte de la etiqueta `<a>`.
- Las URLs pueden ser **absolutas, relativas o internas** (anclas).
- Los enlaces bien escritos deben cuidar la **accesibilidad** (texto claro, sin ambigüedad).

En el plano **SEO**, los enlaces estructuran el sitio y mejoran la indexación si se usan de manera coherente y natural.

Caso de estudio – Wikipedia: el poder del enlace interno

Contexto

Wikipedia es uno de los mejores ejemplos de **arquitectura de enlaces internos**. Cada artículo contiene decenas de hipervínculos que conectan con otros artículos relacionados dentro del propio sitio.

Estrategia

1. **Enlazado contextual:** cada palabra o frase relevante se enlaza a un artículo relacionado, lo que permite al lector ampliar su conocimiento sin salir del sitio.
2. **Estructura jerárquica clara:** los temas principales se enlazan desde categorías o portales (como "Ciencia", "Historia" o "Deporte").
3. **URLs limpias y semánticas:** cada URL incluye palabras descriptivas (/html, /teoria-del-color, /revolucion-industrial).
4. **Accesibilidad:** todos los enlaces son descriptivos y legibles por los lectores de pantalla.
5. **Sin enlaces rotos:** los bots de mantenimiento revisan automáticamente los hipervínculos para garantizar su integridad.

Resultado

- **Mayor tiempo de permanencia** de los usuarios, gracias al flujo de navegación continua.
- **Excelente indexación SEO:** Google entiende las relaciones temáticas entre artículos.
- **Confianza y autoridad:** millones de enlaces externos apuntan a Wikipedia, reforzando su posición en los resultados de búsqueda.
- **Modelo de referencia:** su arquitectura interna es estudiada como estándar de buenas prácticas de SEO técnico.

Wikipedia demuestra que la potencia de la web está en los enlaces bien construidos: conectan conocimiento, mejoran la experiencia y refuerzan la visibilidad.

Herramientas y consejos

Extensiones y validadores

- **WAVE (Web Accessibility Evaluation Tool)**: analiza la accesibilidad de enlaces, botones y textos alternativos.
- **Broken Link Checker**: revisa si hay enlaces rotos dentro de un sitio (versión web o plugin de WordPress).
- **Ahrefs / SEMrush**: analizan los enlaces internos y externos de un sitio, útiles para auditorías SEO.

Buenas prácticas para enlaces

- Escribe anchor text descriptivos: "Ver precios de servicios" es mejor que "Haz clic aquí".
- Añade rel="noopener noreferrer" siempre que uses target="_blank".
- Usa rutas relativas dentro de tu dominio y URLs absolutas solo para recursos externos.
- Evita cadenas de redirecciones (redirigir múltiples veces reduce la autoridad del enlace).
- Coloca los enlaces más importantes cerca del inicio del contenido (Google los prioriza).

Accesibilidad profesional

- Asegúrate de que los enlaces sean accesibles con teclado (Tab y Enter).
- Si el enlace descarga un archivo o abre una nueva pestaña, indícalo en el texto. Ejemplo:

```
<a href="informe.pdf">Descargar informe (PDF)</a>
```

- Evita enlaces vacíos o sin destino (href="#") porque confunden a los usuarios de lectores de pantalla.

SEO on-page aplicado

- Crea una red interna coherente: enlaza artículos o páginas de forma natural y contextual.
- Usa palabras clave en el texto del enlace sin forzarlas (Google penaliza el abuso).
- Controla el número de enlaces salientes: muchos enlaces sin relevancia pueden diluir la autoridad de la página.
- Comprueba con Google Search Console qué páginas reciben más enlaces internos y optimiza las que tienen pocos.

URLs limpias y seguras

- Mantén las URLs cortas y semánticas (ej. /productos/nuevos en lugar de /p?id=123).
- Evita caracteres especiales, mayúsculas o espacios.
- Usa HTTPS siempre; los enlaces a contenido inseguro (http) pueden marcarse como "no seguros" en los navegadores.

Mitos y realidades

✗ **Mito:** "Cuantos más enlaces tenga mi página, mejor posicionará en Google." → **FALSO.** La calidad es más importante que la cantidad. Los enlaces deben ser relevantes y útiles para el usuario. Google valora la coherencia semántica y penaliza el exceso de enlaces irrelevantes (prácticas conocidas como link spamming).

✗ **Mito:** "Las URLs absolutas y relativas son iguales." → **FALSO.** Las URLs relativas son más flexibles dentro de tu propio sitio (si cambias el dominio, siguen funcionando), pero las absolutas son más seguras para enlaces externos o cuando el contenido se comparte fuera del contexto original. La elección depende del uso, no de la apariencia.

❑ Resumen final (para examen)

- `` crea un enlace; href define su destino.
- **URLs absolutas:** enlazan a dominios externos.
- **URLs relativas:** enlazan dentro del mismo sitio.
- **Accesibilidad:** texto del enlace debe ser **descriptivo** y no genérico.
- **SEO:** enlaces internos → mejor indexación; enlaces externos de calidad → mayor autoridad.
- **Buenas prácticas:** usa target=" _blank" con rel="noopener noreferrer", evita enlaces rotos y valida tu sitio con **WAVE** o **Broken Link Checker**.