

## Situación profesional 2: Compaginación HTML

Una vez aceptada la propuesta de parte de nuestro comitente Copan Arquitectura, se comienza con la estructura base en lenguaje HTML para el armado de las vistas de manera estática, siendo esta la etapa 2 del proyecto.

En esta etapa se deberán crear todas páginas HTML, además de las secciones de la SPA que serán renderizadas en el front-end (CSR).

Para el desarrollo se deberá tener en cuenta el diseño estipulado al cuál se deberá desglosar a fin de estructurar correctamente el HTML. Si bien no es definitiva, ya que podrá haber ajustes posteriores sobre esta estructura, se deberán respetar los lineamientos previstos.

1. Detectar la estructura base que se repetirá en todas las vistas, y estructurar un layout común para todas las páginas.
2. Desglosar la interfaz por componentes funcionales mínimos: Encabezado, menú de navegación, pie de página, componentes de información compuestos (imagen, título, texto).
3. Determinar los elementos semánticos a utilizar.
4. Crear los meta elementos necesarios para un buen SEO para cada página.

---

comienza desde la raíz del sistema de archivos, mientras que una ruta relativa se refiere a la ubicación relativa desde la posición actual.

## Lenguaje HTML

Para iniciar el recorrido de los tres lenguajes ya citados, y que son fundamentales a la hora del desarrollo de aplicaciones web del lado del cliente. Desarrollaremos estas herramientas y empezaremos con la base sobre la cual se apoyan los otros lenguajes, estamos hablando del lenguaje HTML.

## Documentación

La costumbre de utilizar la documentación de los lenguajes se hace imprescindible, ya que sería casi imposible memorizar la gran cantidad de información que contiene el lenguaje HTML.

Para AW1 utilizaremos las documentaciones oficiales donde se especifican los estándares, y las creadas por la comunidad de uso frecuente.

En el caso de HTML, disponemos de:

- WHATWG (Web Hypertext Application Technology Working Group) - [Multipage Version](#)
- W3 Consortium - [Web Standards | W3C](#)
- HTML en MDN - [HTML: Lenguaje de etiquetas de hipertexto | MDN](#)
- Además de documentación, manuales, tablas de compatibilidad y recursos prácticos que será anexada a las clases.

## SP2/H1: Herramientas de desarrollo

Para dar inicio al desarrollo Front-end, necesitamos herramientas que nos permitan realizar el trabajo de manera fluida a la hora de codificar, además de trabajar con el resto del equipo.

También, conocer algunos conceptos que debemos manejar para entender mejor el desarrollo de software.

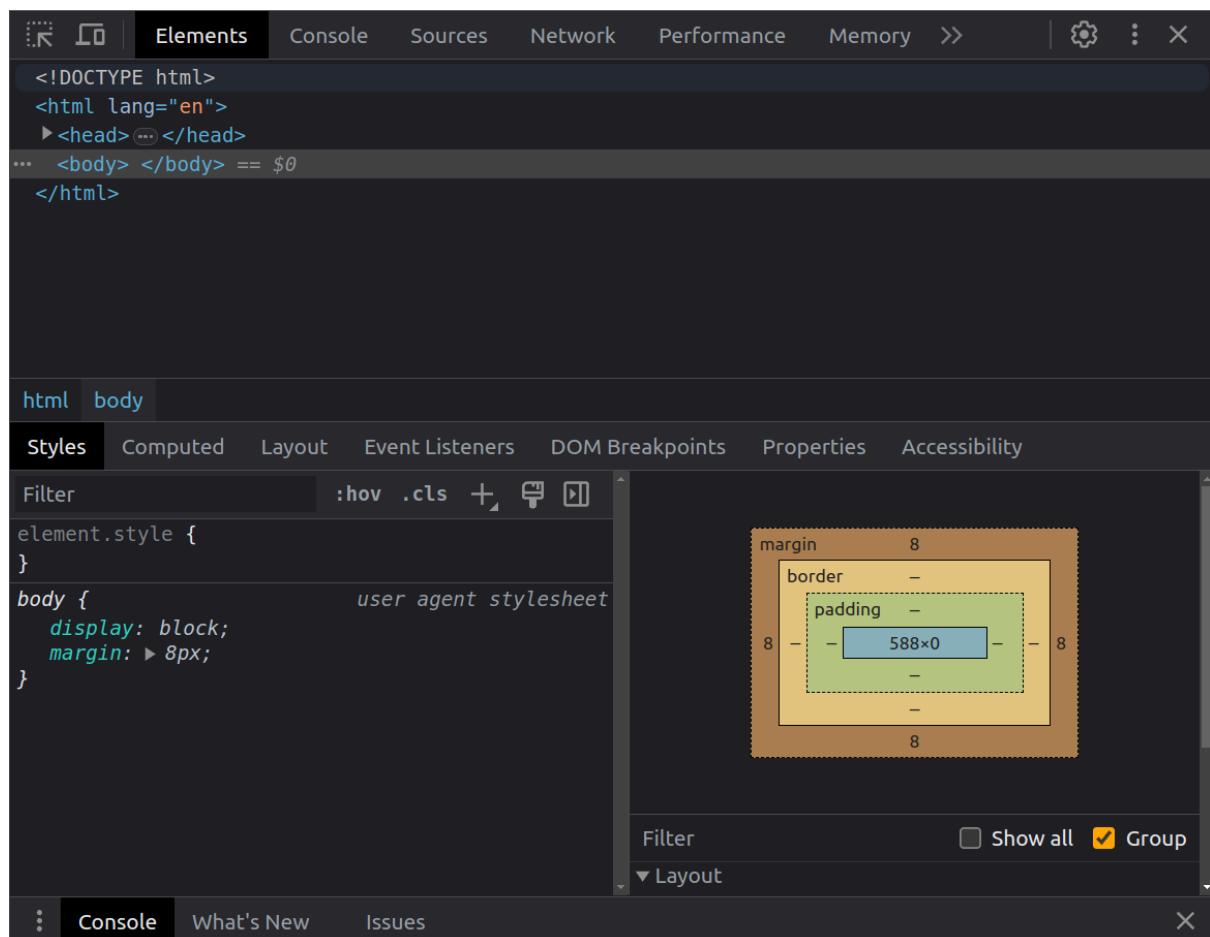
Las herramientas que dispone una persona en el mundo del desarrollo de aplicaciones web son muchas y diversas. Cada equipo y cada empresa adoptan herramientas que permiten agilizar el desarrollo. Además, estas herramientas evolucionan a pasos acelerados, por lo que como parte de nuestro quehacer, debemos estar siempre atentos y atentas a esto.

### Navegador Web o Web Browser

Si bien el navegador se utiliza para el acceso a las aplicaciones web, es un fuerte aliado a la hora de trabajar, ya que todos los navegadores disponen de herramientas que nos permiten visualizar y depurar errores, además de otras funcionalidades.

Estas funcionalidades las encontramos en el menú de cada navegador bajo los nombres de “**inspector**” o “**herramientas de desarrollo**”. En inglés Developer Tools.

Esta es la consola del Navegador Chrome:



## Entorno de desarrollo integrado (IDE)

Todo el desarrollo de una aplicación web lo podemos hacer desde un simple editor de texto y funcionará igual, pero la utilización de un IDE nos facilitará el trabajo.

Un IDE es un entorno de desarrollo integrado (Integrated Development Environment). Es un software de desarrollo que integra varias herramientas y que nos ayuda a trabajar mejor y más rápido.

En los ejemplos verán capturas de pantallas en donde el código tiene distintos colores, esa es una de las ventajas de utilizar un IDE, porque dispone de herramientas como esa que nos ayuda a la lectura del código entre muchas otras

De ahora en más, utilizaremos uno para nuestros desarrollos.

## Creación del entorno personal de trabajo

Es importante a la hora de trabajar poder organizarnos con las herramientas e implementar una metodología consecuente, esto nos permitirá también que el flujo de trabajo sea más rápido.

En primer lugar, conocer nuestras herramientas y sus capacidades. En segundo lugar, decidir qué procedimientos y metodologías vamos a utilizar y ser consecuentes con eso.

Cuando hablamos de metodologías y procedimientos, hablamos de implementar pasos que se organizan y repiten de la misma manera. En consecuencia, podemos comunicar al resto del equipo cómo se va a trabajar.

La mayoría de las veces la organización depende de las lógicas que nos dan las distintas herramientas.

Un ejemplo básico sería el siguiente:

1. Crear un directorio en nuestra computadora con un nombre que lo identifique.
2. Sincronizar nuestro IDE con ese directorio.
3. Inicializar para el proyecto un repositorio de control de versiones.
4. Inicializar y configurar las herramientas para ese proyecto particular y que utilizaremos para proyectos similares.
5. Organizar el proyecto en distintos archivos y directorios.

Dijimos que a veces las herramientas nos organizan las lógicas de trabajo, esto lo veremos más claro cuando seleccionemos el IDE, las herramientas y tipo de proyecto. Normalmente los proyectos se organizan por convenciones, es decir, se utilizan maneras de trabajar ya probadas por otras personas.

## Control de versiones

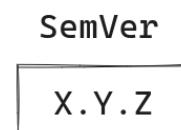
Más arriba nombramos los conceptos de “repositorio” y de “control de versiones”. Estos conceptos están relacionados, porque un repositorio es un lugar físico, como un directorio en nuestra computadora o en la nube, que será el encargado de almacenar y catalogar las distintas versiones de nuestro código.

## Software de control de versiones

Utilizar un control de versiones es una buena práctica. Para ello existen herramientas dedicadas a este propósito. Una de ellas, y la más extendida, es **GIT**<sup>21</sup>, un software de código abierto y libre que fue creado por Linus Torvalds, creador del sistema operativo Linux, en el 2005.

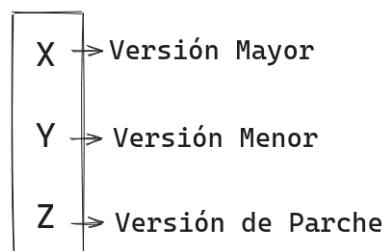
## Nomenclatura de versionado

Por otra parte, debemos adoptar una nomenclatura de versionado<sup>22</sup> para organizar el código. En nuestro caso adoptaremos para esta materia la nomenclatura de versionado semántico<sup>23</sup> o **SemVer** que tiene este formato:



Donde X, Y y Z son números enteros no negativos; no deben estar precedidos de cero, salvo que sea solo cero, y se deben incrementar numéricamente.

X es la versión Mayor, Y es la versión Menor y Z es la versión de Parche.



## Ejemplo

Supongamos que lanzamos nuestra primera versión en producción de una aplicación y le asignamos la numeración **1.0.0**. Luego de un tiempo de uso por parte de los usuarios, se detecta un error o **bug**, como equipo de desarrollo lo corregimos y lanzamos a producción una nueva versión, ¿cómo quedaría esta nueva numeración?, así: **1.0.1**

---

<sup>21</sup> [Git](#)

<sup>22</sup> [Versionado de software - Wikipedia, la enciclopedia libre](#)

<sup>23</sup> [Versionado Semántico 2.0.0 | Semantic Versioning](#)

Pasado un tiempo del lanzamiento de la aplicación nos damos cuenta de que necesitamos agregar una nueva funcionalidad (feature<sup>24</sup>), es decir, algo que mejora el software, pero que no es un error, una vez implementada la liberamos con el número **1.1.1**

Ahora bien, ¿cuándo cambiamos la versión mayor?, la versión mayor solo es cambiada cuando es necesario romper la compatibilidad con una versión anterior, es decir, hay un cambio en el corazón (core) y se reescriben funcionalidades o estas funcionalidades cambian lo que hacían en la versión anterior, lo que no permite esa compatibilidad. Una versión **1.2.25** no es compatible con una versión **2.0.2**

## El uso de las documentaciones

Otra herramienta fundamental es la utilización de la documentación, en ella encontraremos el correcto uso de las herramientas. Debemos acostumbrarnos a consultarlas frecuentemente y serán parte de esta configuración o setup del entorno de desarrollo.

---

<sup>24</sup> Aparece normalmente en inglés aunque estemos hablando en castellano.

### Elija la opción correcta

1 - ¿Qué es un IDE en el desarrollo de aplicaciones web?

- a) Un lenguaje de programación.
- b) Un entorno de desarrollo integrado.
- c) Una metodología de desarrollo.

2 - ¿A qué nos referimos con "Dev Tools" o "Herramientas de Desarrollo" en el Navegador Web?

- a) Herramientas de desarrollo y depuración externas al Navegador Web.
- b) Herramientas de desarrollo y depuración instalables en el Navegador Web.
- c) Herramientas de desarrollo y depuración propias del Navegador Web.

3 - ¿Qué es un repositorio de control de versiones?

- a) Un lugar físico para desarrollar aplicaciones.
- b) Un lugar físico para almacenar y catalogar distintas versiones de código.
- c) Un entorno físico de desarrollo integrado para controlar el código.

4 - ¿Cuál de las siguientes es una herramienta de control de versiones ampliamente utilizada en el desarrollo de aplicaciones web?

- a) FTP.
- b) HTTP.
- c) Git.

5 - ¿Qué es la nomenclatura de versionado semántico o SemVer?

- a) Una forma de organizar archivos en directorios.
- b) Una convención para asignar números de versión a software.
- c) Una técnica para depurar código.

6 - En la nomenclatura SemVer, ¿qué representa la versión Mayor?

- a) Una nueva versión de “ parche” incompatible con la versión anterior.
- b) Una nueva versión que “ rompe” compatibilidad con la versión anterior.
- c) Una nueva versión de “ mejora” compatible con la anterior.

7 - En un versionado SemVer supongamos que una aplicación se lanza con la versión 1.0.0 y se corrige o crea un parche solucionando un error o bug. ¿Cuál sería la nueva versión?

- a) 0.1.0
- b) 1.0.1
- c) 1.1.0

Opciones correctas<sup>25</sup>

---

<sup>25</sup> 1.b) Un entorno de desarrollo integrado. 2. c) Herramientas de desarrollo y depuración propias del Navegador Web. 3. b) Un lugar físico para almacenar y catalogar distintas versiones de código. 4. c) Git. 5. b) Una convención para asignar números de versión a software. 6. b) Una nueva versión que “ rompe” compatibilidad con la versión anterior. 7. b) 1.0.1

## SP2/H2: Definición y Sintaxis

### Definición

HTML es el lenguaje más básico de la Web que define el **significado** y **estructura** de una página web. **Significado**, porque a través de marcas da sentido a los contenidos, y **estructura**, porque organiza y jerarquiza esos contenidos. **La última versión de este lenguaje es la 5, y data del 2014.**

La palabra **Hipertexto** hace referencia a que existen enlaces o vínculos (hipervínculos) que conectan otras páginas y recursos, así como enlaces dentro de la misma página, de esta manera podemos “navegar” o saltar de un documento a otro o de un lugar a otro.

Su sintaxis está construida a partir de “etiquetas” que funcionan como “marcas” las cuales son interpretadas por el Navegador Web para representar o renderizar el contenido de manera organizada.

El marcado se realiza en texto plano conformando el **documento HTML**, y la extensión que lleva ese archivo es **.html**



### Sintaxis

Si el documento se escribe en texto plano, ¿cómo se realiza el marcado para que el Navegador lo entienda? Allí es donde entra el concepto de marcado, que son etiquetas escritas con caracteres comunes con una sintaxis especial.

## Etiqueta

Básicamente las marcas se reconocen porque llevan estos dos símbolos < (menor que) y > (mayor que) que contienen un nombre, el nombre de la etiqueta.

**<etiqueta>**

Estas etiquetas se utilizan para “encerrar” un contenido para que el navegador pueda interpretar ese contenido y darle el tratamiento que corresponda.

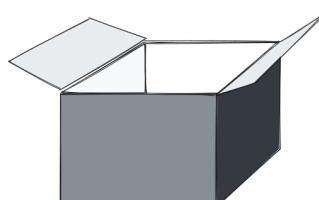
**<etiqueta> Contenido </etiqueta>**

*¡La etiqueta por defecto<sup>26</sup> <etiqueta> no existe!, es solo a modo de ejemplo. Las etiquetas tienen nombres muy específicos que ya veremos.*

A su vez, estas etiquetas pueden encerrar otras etiquetas en su interior:

```
<etiqueta>
    <etiqueta>
        Contenido 2
    </etiqueta>
</etiqueta>
```

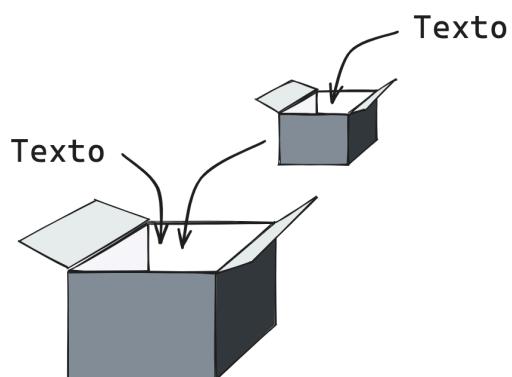
Pensemos esto como cajas:



---

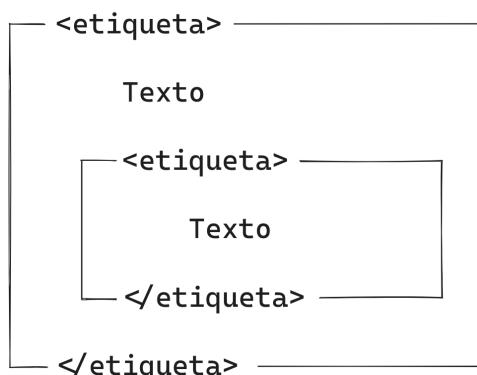
<sup>26</sup> Que vienen incluidas en el lenguaje. Decimos por defecto, porque podemos crear elementos con etiquetas nuevas con JavaScript.

estas cajas pueden almacenar texto u otras cajas:



*Es importante la palabra caja, ya que, cuando veamos el lenguaje CSS, vamos a utilizar este concepto en inglés: Box.*

O pensar así:



Hay etiquetas que no contienen nada, es decir, que no tienen una etiqueta de cierre, y se utilizan normalmente para referenciar recursos, por ejemplo, una imagen.

Se escriben con una barra inclinada antes del cierre así:

```
<etiqueta />
```

También es válido sin la barra inclinada:

```
<etiqueta>
```

Pero esta última forma no es recomendable por cuestiones de legibilidad de otras personas. Recordemos que en esta disciplina trabajamos muchas veces en equipo.

## Atributos

Ya tenemos en claro cómo es la sintaxis de las etiquetas, pero podemos dar más información al Navegador para que entienda cómo debe comportarse una etiqueta. Para eso disponemos de los **atributos** que podemos incluir de esta manera:

```
<etiqueta atributo1="valor" atributo2="valor">  
    Contenido  
</etiqueta>
```

Entonces su sintaxis sería **atributo = valor**

## Comentarios

En la sintaxis HTML podemos incluir comentarios, es decir, texto que el Navegador va a ignorar a la hora de la representación de la página en pantalla, y la sintaxis es la siguiente

```
<!-- Contenido comentado -->
```

## El elemento HTML

El **elemento HTML** es un concepto que surge a partir de la utilización del lenguaje JavaScript para comunicarse con el documento y no debe confundirse con la etiqueta. Este **elemento** es una entidad completa y se compone de:

- Las etiquetas.
- Los atributos.

- El contenido.

```
<p id="idDelElemento" class="claseCSS">  
    Párrafo  
</p>
```

No vamos a enumerar aquí todos los elementos, ya que nos llevaría un apunte anexo, solo vamos a utilizar algunos a fin de exemplificar y practicar.

*Como desarrolladores y desarrolladoras, debemos revisar la documentación constantemente.*

Aquí está la lista completa en dos fuentes externas que pueden consultar:

- WHATWG - Elementos HTML [Multipage Version](#)
- MDN - Elementos HTML [Referencia de Elementos HTML - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

Además, podemos probar directamente online con algunas herramientas:

- MDN Play - [MDN Web Docs](#)
- W3Schools - [HTML Tutorial](#)

## Atributo especial *data-\**

Antes de avanzar, vamos a hablar del atributo especial *data-\**. Este atributo permite almacenar datos en un elemento HTML a partir de una atributo personalizado que puede luego utilizarse como información extra desde CSS y JavaScript:

```
<p data-publicado="10-05-2022">  
    Párrafo  
</p>
```

En el ejemplo de arriba agregamos un atributo data con un nombre descriptivo “publicado” (**data-publicado**). Aquí el uso que se le da es para almacenar una fecha de publicación del texto.

## Comportamiento de línea y de bloque

En HTML hay una variedad importante de elementos que tienen funciones diversas, y además, dos comportamientos a la hora de ser renderizados en el flujo del documento HTML.

*Cuando hablamos de renderizar, hablamos de la interpretación que hace el Navegador de toda la información contenida en el documento HTML para mostrar en pantalla el resultado.*

Los **dos** comportamientos de los elementos HTML a la hora de ser renderizados, y que debemos entender, son:

- el elemento que se comporta como **bloque**.
- el elemento que se comporta como **línea**.

Estos comportamientos los veremos más en profundidad en la sección del lenguaje CSS.

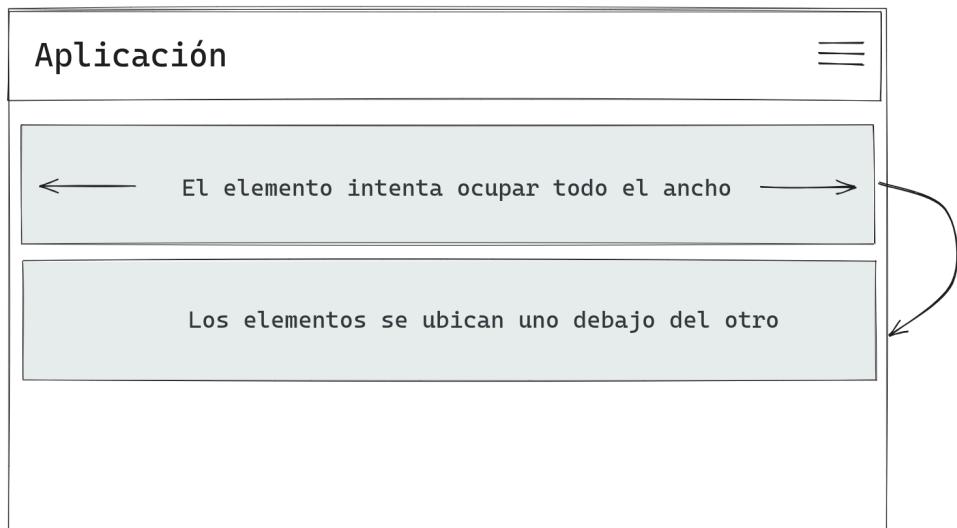
**IMPORTANTE:** *Por norma, nunca debemos incluir un elemento de bloque dentro de uno de línea.*

Documentación:

- [Elementos en bloque - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web](#)
- [https://developer.mozilla.org/es/docs/Orphaned/Web/HTML/Inline\\_elements](https://developer.mozilla.org/es/docs/Orphaned/Web/HTML/Inline_elements)

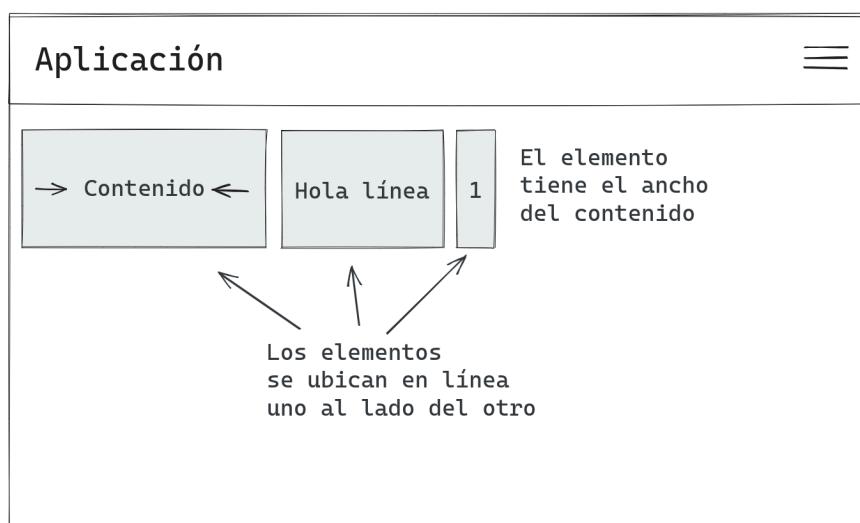
## Comportamiento de bloque

Los elementos de este tipo, intentarán ocupar todo el ancho disponible y forzará al salto de línea, es decir, cada elemento nuevo estará ubicado debajo del anterior, no importa si el anterior es de bloque o de línea:



## Comportamiento de línea

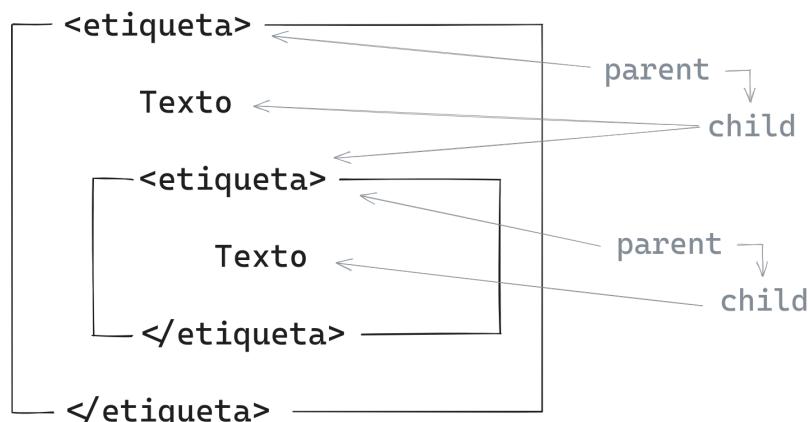
Los elementos de este tipo tendrán el tamaño del contenido y estarán uno al lado del otro hasta que lleguen al ancho disponible, luego se hará un salto de línea y se ubicarán por debajo.



## Anidación *parent* y *child*

Más arriba hablamos de que podemos pensar las etiquetas como cajas que pueden contener otras en su interior. Continuando con esa lógica, y ahora que vimos el concepto de **elemento**, podemos decir que si un elemento “contiene” a otro elemento, le llamaremos **anidación**, esto permite agrupar y jerarquizar los contenidos dentro de un documento HTML. En la anidación, el elemento **contenedor** se denomina **parent** (ancestro), y los elementos contenidos serán **children** (descendientes).

Nos referiremos a ellos dependiendo el contexto, ya que un elemento parent puede a su vez ser un child de otro elemento:



Recordemos que siempre utilizamos los términos en inglés, ya que día a día lo veremos expresado en ese idioma, además, nos va a facilitar la comprensión cuando interactuemos entre los distintos lenguajes.

### Elija la opción correcta

1. HTML es

- a) Un lenguaje de programación.
- b) Un lenguaje de marcado.
- c) Un lenguaje de hipervínculo.

2. ¿Cómo reconocer un Navegador Web el marcado HTML?

- a) Por sus etiquetas.
- b) Por sus saltos de línea.
- c) Por la extensión del archivo HTML

3. Los atributos se escriben:

- a) <etiqueta> Texto </etiqueta atributo="valor">
- b) <etiqueta> atributo="valor" </etiqueta>
- c) <etiqueta atributo="valor"> Texto </etiqueta>

4. Los comentarios se escriben:

- a) <- Este es un comentario ->
- b) <! Este es un comentario >
- c) <!-- Este es un comentario -->

5. Un elemento HTML está compuesto por:

- a) Por sus etiquetas, contenidos y comentarios.
- b) Por sus etiquetas, atributos y contenido.
- c) Por sus atributos, textos y contenido.

6. Hay dos comportamientos de los elementos HTML dentro de su flujo normal, estos son:

- a) De bloque y Flotante.
- b) Estático y de línea.
- c) De bloque y de línea.

7. Un comportamiento de bloque:

- a) Tiende a ocupar el ancho disponible y obliga al salto de línea.
- b) Sólo ocupa el ancho del contenido y obliga al salto de línea.
- c) Tiende a ocupar el ancho disponible y no obliga al salto de línea.

8. La anidación de elementos HTML se da cuando:

- a) Un elemento HTML está al lado del otro elemento HTML
- b) Un elemento HTML está debajo de otro elemento HTML
- c) Un elemento HTML está dentro de otro elemento HTML

### Opciones correctas<sup>27</sup>

---

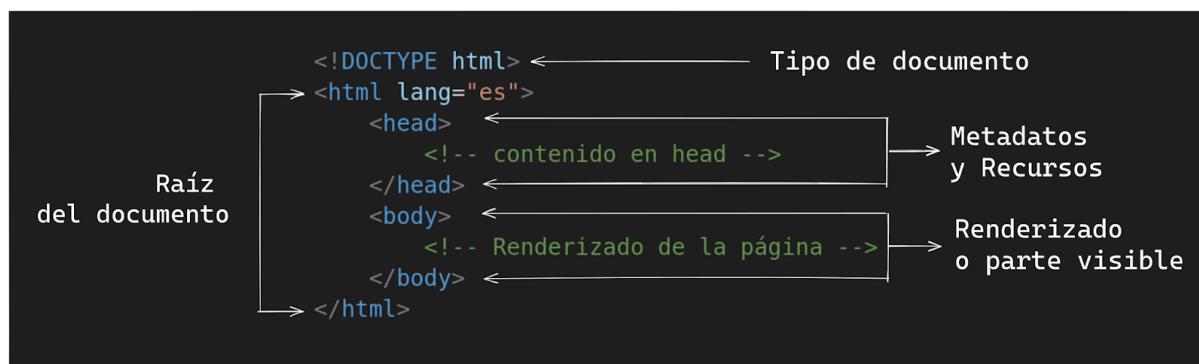
<sup>27</sup> 1.b) Un lenguaje de marcado. 2. a) Por sus etiquetas. 3. c) <etiqueta atributo="valor"> Texto </etiqueta>. 4. c) <!-- Este es un comentario -->. 5. b) Por sus etiquetas, atributos y contenido. 6. c) De bloque y de línea. 7. a) Tiende a ocupar el ancho disponible y obliga al salto de línea. 8. c) Un elemento HTML está dentro de otro elemento HTML.

## SP2/H3: El documento HTML

Ahora, con los conceptos de la sintaxis claros, abordaremos la manera en que se estructura un documento HTML para que el Navegador pueda interpretarlo correctamente.

En esta herramienta veremos cómo se compone un documento HTML.

### Estructura base



`<!DOCTYPE html>`

Es la etiqueta que le dice al navegador que se trata de un documento HTML

`<html></html>`

Es la etiqueta principal y raíz o **root** en inglés. La más externa del documento. Esta etiqueta debería tener siempre el atributo `lang`, que hace referencia al idioma utilizado por el documento: `<html lang="es">`

`<head></head>`

Es la etiqueta en donde indicamos metadatos, es decir, información necesaria para el correcto renderizado<sup>28</sup>. Además, enlaces a recursos que utiliza el documento como Hojas de estilo, Script, entre otros.

`<body></body>`

<sup>28</sup> El renderizado es la interpretación y representación de la página en el navegador.

Es la parte visible de la página. Es donde se lleva a cabo el renderizado propiamente dicho.

## Head

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <!-- contenido en head -->

        <!-- meta -->
        <meta charset="UTF-8" />
        <meta name="description" content="Mi descripción" />

        <!-- link -->
        <link rel="stylesheet" href="style.css" />
        <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />

        <!-- título -->
        <title>El head</title>
    </head>
    <body>
        <!-- Renderizado de la página -->
    </body>
</html>
```

Como dijimos, en el head vamos a encontrar varias etiquetas y sus atributos que dan información extra al Navegador:

- al conjunto de caracteres a utilizar `<meta charset="utf-8" />`
- a una descripción `<meta name="description" content="Mi descripción" />`
- al título `<title>Mi título</title>`

Linkear recursos externos:

- Link a hoja de estilo `<link rel="stylesheet" href="style.css" />`
- Link a un ícono para la pestaña del navegador `<link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />`

Título del documento:

- <title>Mi página HTML</title> esta información aparece en la pestaña del Navegador.

## Body

En el body incluiremos todos los elementos que serán renderizados.

```
<!DOCTYPE html>
<html lang="es">
  <head>
    <!-- contenido en head -->

    <!-- meta -->
    <meta charset="UTF-8" />
    <meta name="description" content="Mi descripción" />

    <!-- link -->
    <link rel="stylesheet" href="style.css" />
    <link rel="shortcut icon" href="favicon.ico" type="image/x-icon" />

    <!-- título -->
    <title>El head</title>
  </head>
  <body>
    <!-- Renderizado de la página -->
    <h1>Título más importante</h1>
    <p>
      Texto de un párrafo
    </p>
    
    <h2>Título menos importante</h2>
    <p>
      Texto de un párrafo
    </p>
  </body>
</html>
```

Esto en el navegador se ve así:

## Título más importante

← <h1>

Texto de un párrafo ←

<p>

```
/*  
 * @var boolean  
 */  
define('PSI_INTERNAL_XML', false);  
if (version_compare("5.2", PHP_VERSION, ">")) {  
    die("PHP 5.2 or greater is required!!!");  
}  
if (!extension_loaded("pcre")) {  
    die("phpSysInfo requires the pcre extension to php in order to work  
        properly.");  
}  
  
require_once APP_ROOT . '/includes/autoload.inc.php';  
  
// Load configuration  
require_once APP_ROOT . '/config.php';  
if (!defined('PSI_CONFIG_FILE')) || !defined('PSI_DEBUG')) {  
    $tpl = new Template("/templates/html/error_config.html");  
    echo $tpl->fetch();  
    die();  
}  
... javascript  
... strToLower(
```

← <img />

## Título menos importante

← <h1>

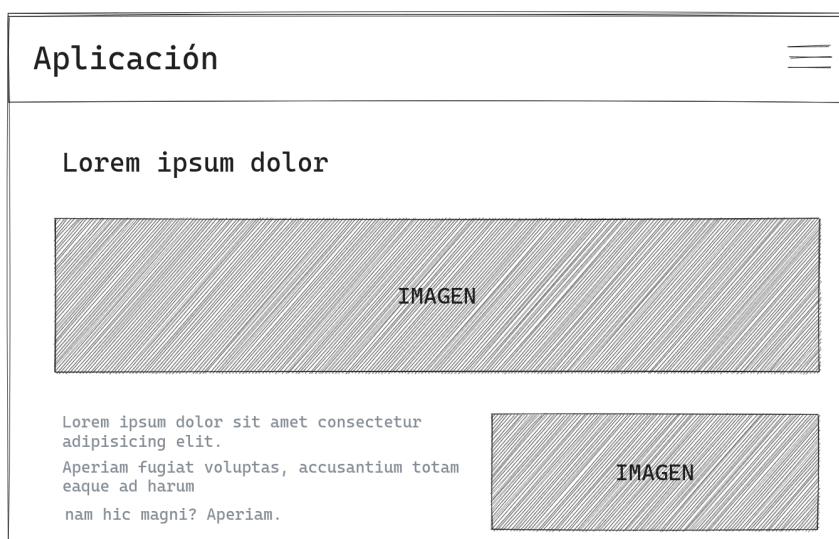
Texto de un párrafo ←

<p>

Como observamos, el Navegador agrega tamaños y tipos de letras por defecto dependiendo el elemento HTML que utilicemos, además de rellenos y márgenes. Todo esto se podrá modificar utilizando el lenguaje de Hojas de Estilo en Cascada o CSS.

## Layout o Compaginación

Un término en inglés muy utilizado para referirse a una estructura y disposición de los elementos de página web es el **layout**, que se puede traducir como **compaginación**. El **layout** hace referencia a la estructura del documento y la disposición de los contenidos a vincular y renderizar que fue decidida por el diseñador o diseñadora web.



Veamos el código fuente de una página web cuya manera de ordenar los contenidos la llamaremos **layout**:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <title>AW1 | IES</title>
    </head>
    <body>
        <header>
            <h1>Aplicaciones web</h1>
            <nav>
                <a href="inicio.html">Inicio</a> <a href="pagina2.html">Otra página</a>
            </nav>
        </header>
        <main>
            <section>
                <h2>Título</h2>
                <p>
                    Lorem ipsum dolor sit amet consectetur adipisicing elit.
                    Aperiam fugiat voluptas, accusantium totam eaque ad harum
                    nam hic magni? Aperiam.
                </p>
                <ul>
                    <li>Accusantium</li>
                    <li>Lorem ipsum</li>
                </ul>
            </section>
        </main>
        <footer>
            <h4>Footer</h4>
            <p>
                Lorem ipsum dolor sit, amet consectetur adipisicing elit.
                Suscipit, esse?
            </p>
        </footer>
    </body>
</html>
```

Su renderizado en el Navegador se verá algo así:

# Aplicaciones web

[Inicio](#) [Otra página](#)

## Titulo

Lorem ipsum dolor sit amet consectetur adipisicing elit. Aperiam fugiat voluptas, accusantium totam eaque ad harum nam hic magni? Aperiam.

- Accusantium
- Lorem ipsum

## Footer

Lorem ipsum dolor sit, amet consectetur adipisicing elit. Suscipit, esse?

Describamos algunos elementos:

`<header></header>`

Con estas etiquetas indicamos un elemento de encabezado.

`<nav></nav>`

Con `nav` indicamos que es una barra de navegación a distintas páginas. Dentro de `<nav>` estarán los enlaces o anclas.

`<a></a>`

Anclas o vínculos, de los cuales profundizaremos luego en un apartado.

`<h1></h1>`

Indica un título. Existen seis niveles, desde el `<h1>` que es el más importante, y de forma descendente al `<h6>`, el menos importante.

`<main></main>`

Es el contenido principal de la página y solo debe haber un solo elemento `<main>`.

**<section></section>**

Demarca una sección genérica de contenido.

**<p></p>**

Demarca un párrafo de texto.

**<ul></ul> y <li></li>**

Es una lista desordenada y hay que indicar los ítems de la lista con **<li>**. Para que muestre números como una lista ordenada, debemos utilizar la etiqueta **<ol>** en lugar de **<ul>**

**<footer></footer>**

Sección de pie de página o de sección.

Cuando una página se renderiza muchos de los elementos se muestran como bloques de los cuales no podemos identificar a simple vista, salvo que veamos el código fuente, pero que es muy importante saber qué función cumple cada elemento dentro de la estructura, en otras palabras, si tenemos que indicar un título, utilizaremos la familia de los títulos **<h1>** al **<h6>**; si tenemos que indicar un encabezado, utilizaremos **<header>**; si tenemos que indicar una sección, utilizaremos **<section>**; y así sucesivamente.

A continuación veremos por qué es importante respetar el **significado** de cada elemento utilizado, a esto se llama marcado semántico.

## Marcado semántico

A partir de la versión HTML5 se incluyeron una serie de elementos que permiten entender mejor la estructura de una página web. Estos elementos se denominan elementos semánticos, y al utilizarlos, van dando significado o roles a los contenidos.

Podemos añadir dos razones más que son fundamentales para utilizar este marcado, por un lado, torna a un documento HTML más accesible para los lectores de pantalla, y por otro, los buscadores pueden indexar mejor los contenidos.

Aquí solo algunos de los elementos semánticos agregados en la versión 5 de HTML para estructurar los contenidos:

- <header>
- <nav>
- <section>
- <article>
- <aside>
- <footer>
- <figure>

Más adelante ampliaremos más sobre la accesibilidad y el marcado semántico.

En el apartado anterior describimos algunos elementos de una página web y podemos observar a simple vista, qué funciones o roles tienen los distintos bloques con solo leer la etiqueta de cada elemento.

Hay dos razones fundamentales para este marcado, por un lado torna a un documento HTML más accesible para los lectores de pantalla, y por otro, los buscadores pueden indexar mejor los contenidos.

Más adelante ampliaremos más sobre la accesibilidad y el marcado semántico.

## El elemento ancla o *anchor*

La definición de HTML contiene la palabra Hipertexto<sup>29</sup>, que es la manera de definir documentos que están enlazados a partes del mismo documento o a otros documentos a partir de **hipervínculos** o **hiperenlaces**, conformando una red de referencias a contenidos.

---

<sup>29</sup> <https://es.wikipedia.org/wiki/Hipertexto>

Un hipervínculo, o su forma más utilizada en inglés **Link** (vínculo), es una referencia unidireccional hacia sí mismo u otro documento que puede estar en cualquier parte de la Web, es decir, en el mismo servidor del documento actual, o en otros servidores.

También debemos decir que es un elemento interactivo o activo el cuál el usuario debe presionar para dirigirse a la sección o documento referenciado.

Para crear un vínculo, HTML nos provee del elemento **anchor <a>**

```
<a href="index.html">Ir a la página de inicio</a>
```

Su atributo más importante, y que indica el destino del link es **href**, y como vemos, envuelve un texto que es el texto que veremos renderizado. Por defecto, el Navegador lo colorea de azul y lo subraya:

[Ir a la página de inicio](#)

Ver documentación:

<https://developer.mozilla.org/es/docs/Web/HTML/Element/a>

## Atributos **target** y **download**

Existen otros dos atributos de uso frecuente que nos permiten cambiar el comportamiento:

- **target**

Con el valor \_blank indicamos que se abra el recurso en una pestaña nueva.

```
<a href="https://dominio.com/" target="_blank">Vamos</a>
```

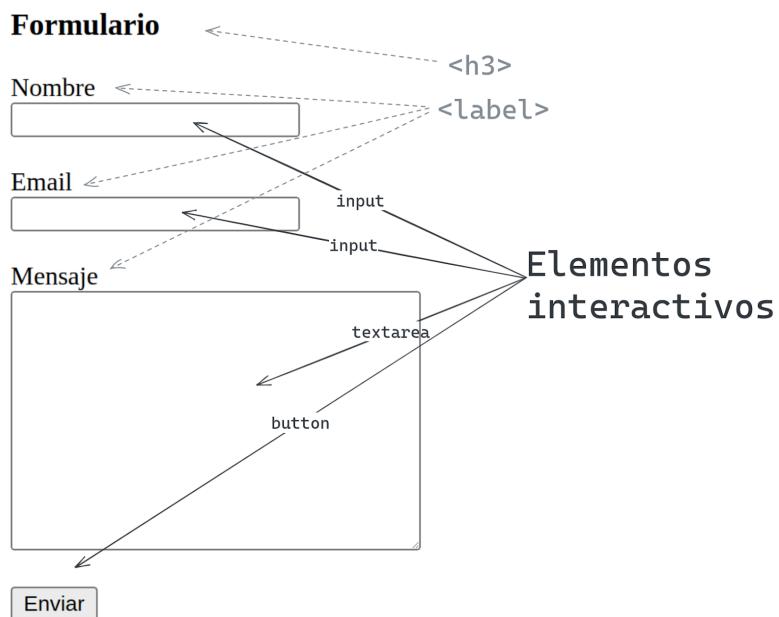
- **download**

Indica que el recurso es de descarga. No hace falta especificar un valor, por defecto es **true**

```
<a href="https://articulos.com/articulo.pdf" download>Descarga del PDF</a>
```

## El elemento formulario

El elemento de formulario o **form**, más los **componentes interactivos** que permiten a los usuarios introducir datos en una página web, es la manera que tiene HTML por defecto de enviar datos a un servidor.



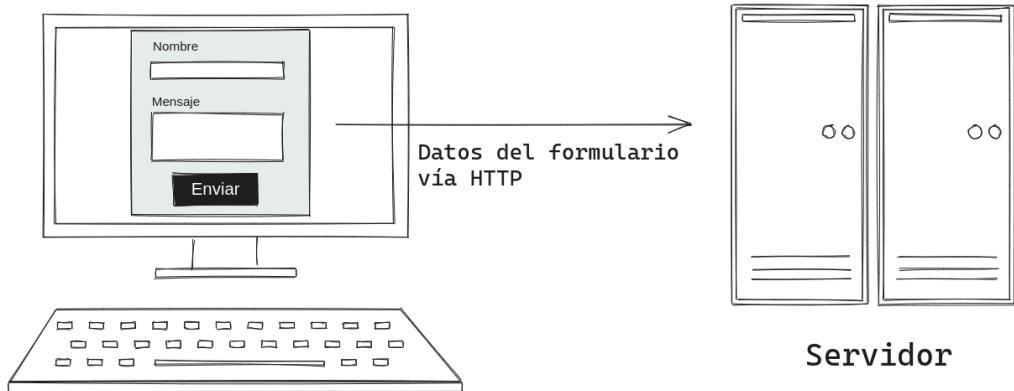
## Funcionamiento

Utilizando solamente HTML puro<sup>30</sup>, el funcionamiento básico de un formulario es el siguiente:

1. Un usuario ingresa los datos en el formulario, es decir, rellena los campos disponibles.
2. Acciona un botón que indica que los datos se deben enviar al servidor.
3. Indica al Navegador Web que inicie la transferencia de datos utilizando el protocolo HTTP.

---

<sup>30</sup> Decimos puro, porque no se necesita nada más. Más adelante veremos cómo podemos utilizar JavaScript para hacer lo mismo.



## Estructura

La etiqueta de marcado que se utiliza para indicar la presencia de un formulario es `<form>`<sup>31</sup>

```
<form>
    <!-- Componentes interactivos -->
</form>
```

## Atributos *action* y *method* del form

Ya estuvimos hablando de los atributos, y que éstos dan más información al Navegador para saber cómo debe tratarlos.

En el caso del formulario, existen dos atributos que son los que siempre deben estar para el correcto funcionamiento. Estos son el **action** y el **method**.

```
<form action="" method=""
    <!-- Componentes interactivos -->
</form>
```

### **action**

Este atributo indica al formulario cual es el **URL** al cual debe enviar los datos. Ese recurso ubicado en el **URL** será un programa del lado del servidor que procese estos

---

<sup>31</sup> [form - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

datos.

### method

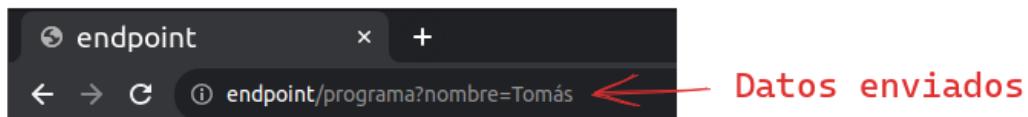
Este atributo indica al formulario el tipo de **método** que debe utilizar a la hora de enviar los datos. Ya hemos enumerado algunos de los métodos disponibles, pero en este caso, solo serán posibles dos métodos de petición:

Con el método **get**, los datos se adjuntarán al URL como **query string** con formato de par clave=valor cuando enviemos el formulario.

```
https://endpoint/programa?clave=valor ←
```

Este **query** es la parte del esquema de un **URI** que aparecerá luego del símbolo ?, y los valores provienen de los componentes interactivos que veremos luego.

```
<form action="https://endpoint/programa" method="get">
    <!-- Componentes interactivos -->
</form>
```



Por otra parte, cuando utilizamos el método **post**, los datos se adjuntan al **body** de la petición.

*Volvamos a Protocolos y repasemos el protocolo HTTP y la estructura de la petición.*

```
<form action="https://endpoint/programa" method="post">
    <!-- Componentes interactivos -->
</form>
```

Para poder ver los datos enviados, podemos utilizar la herramienta de inspección del Navegador Web:

The screenshot shows the Network tab in the Chrome DevTools developer console. A red arrow points from the text "No throttling" in the top toolbar to the timeline, indicating that the request was sent without throttling. Another red arrow points from the text "Datos enviados" to the "Payload" section of the request details, which displays the form data "nombre: Tomás".

### Atributo `name`

Otro aspecto fundamental es definir para cada elemento HTML del formulario un atributo que permitirá dar “nombre” a ese componente el cual será de suma importancia si queremos capturar los valores del lado del servidor enviados desde este formulario.

Este atributo es el `name`. Se utilizará como clave para poder recuperar el valor introducido en este campo por el usuario cuando se envíen los datos al servidor.

```
<input type="text" name="nombre" />
```

↑  
Especifica un nombre único  
para el formulario que lo contiene.

Esto lo adelantamos cuando hablamos sobre la utilización de los métodos de envío.

En el caso del método `get`:



Lo mismo para **post**, pero los datos viajan por el **body** de la petición HTTP.

### Envío del formulario o Submit

Por último, diremos que para que se envíen los datos es necesario un **botón** que accione el envío conocido como submit.

```
<form action="https://endpoint/programa" method="post">
    <!-- componentes del formulario -->
    <button type="submit">Enviar</button>
</form>
```

Botón submit

### Elementos interactivos

Conjuntamente con el elemento **form**, debemos decidir, dependiendo de cómo queremos armar la interfaz del formulario, qué elementos HTML interactivos incluir para permitir la introducción de datos a la página web.

Estos elementos interactivos deberán estar dentro del elemento **<form>**, y como ya vimos, a esto se le llama anidación. De esta manera, los elementos HTML anidados o descendientes de **<form>**, responderán al formulario y todo funcionará como un solo bloque.

*Es importante aclarar que a los elementos del formulario también se los llama de manera genérica "campos".*

Aquí citaremos solo algunos elementos de una gran lista, pero es importante revisar constantemente la documentación que nos dirá cómo debemos utilizarlos, para qué, y sus atributos específicos.

## <input />

El elemento input es un elemento genérico y el más complejo cuya función es definida por el atributo **type**.

```
<input type="text" name="nombre" />
```

Especifica su comportamiento

Aquí hay algunos tipos para asignar al atributo **type**:

(Para la lista completa y todos sus atributos ver la documentación<sup>32)</sup>

- **text**

Se comportará como un campo genérico de texto de una sola línea.

```
<input type="text" name="nombre-del-campo" id="id-del-campo" />
```

Texto de una sola línea |

- **number**

Solo se podrán ingresar números.

```
<input type="number" name="nombre-del-campo" id="id-del-campo" />
```

Se renderiza:

123456 | ▾

---

<sup>32</sup> [input - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

- **email**

Su aspecto sigue siendo como un campo genérico, pero solo se podrá ingresar un email con formato válido.

```
<input type="email" name="nombre-del-campo" id="id-del-campo" />
```

Se renderiza:



hola@mail.com

- **password**

Los caracteres ingresados se representarán como puntos que añaden un nivel de seguridad visual al no poder verse lo que se está ingresando.

```
<input type="password" name="nombre-del-campo" id="id-del-campo" />
```

Se renderiza:



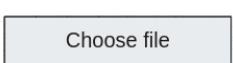
• • • • • • • |

- **file**

se utiliza para el envío de archivos

```
<input type="file" name="nombre-del-campo" id="id-del-campo" />
```

Se renderiza:



Choose file      No file chosen

- **hidden**

es un elemento que el navegador no renderiza y se utiliza para enviar datos extras sin que el usuario pueda manipularlos directamente desde la interfaz.

```
<input type="hidden" name="nombre" value="valor_oculto" />
```

- **checkbox**

Este elemento se utiliza como opción de selección de dos estados: si se selecciona toma el valor del atributo value, si no, no se envía nada.

```
<input type="checkbox" value="un_valor" />
```

Se renderiza:



- **radio**

En este elemento se debe utilizar un mismo nombre en el atributo name, de esta manera funcionan como grupo y solo uno puede ser seleccionado.

```
<input type="radio" name="mismo-nombre" id="id-del-campo-1" value="valor_1" />
<input type="radio" name="mismo-nombre" id="id-del-campo-2" value="valor_2" />
<input type="radio" name="mismo-nombre" id="id-del-campo-3" value="valor_3" />
```

↑  
Mismo nombre

↑  
Distintos valores

Se renderiza:



- **button**

se renderiza como botón genérico.

```
<input type="button" value="Botón" />
```

Se renderiza:

Botón

- **submit**

se renderiza como botón y es el encargado de enviar el formulario. Se renderiza igual al anterior, pero cambia su función.

```
<input type="submit" value="Enviar" />
```

Se renderiza:

Enviar

```
<textarea></textarea>
```

El elemento **<textarea>** nos permite introducir texto multilínea. Los atributos **cols** y **rows** han de referencia al tamaño del campo en el momento del renderizado.

```
<textarea name="nombre-del-campo" id="id-del-campo" cols="30" rows="10"></textarea>
```

Se renderiza:

Campo de área de texto  
Podemos escribir  
- texto  
- multi  
- línea

```
<button></button>
```

El elemento **button** es una alternativa semántica al **<input type="button" />** o **<input type="submit" />**, y que tiene etiqueta de apertura y cierre. También podemos determinar su tipo para cambiar su comportamiento con el atributo **type** **<button type="submit">Mi botón</button>**

```
<button>Mi botón con "Button"</button>
```

Se renderiza:

Mi botón con "Button"

## <select></select>

Es un selector desplegable en el cual podemos seleccionar una opción, o, si está especificado, múltiple opciones. Este es un elemento compuesto que anida otro elemento que representa las opciones con la etiqueta <option>

```
<select name="nombre-selector" id="id-selector">
    <option value="valor-1">Valor 1</option>
    <option value="valor-2">Valor 2</option>
    <option value="valor-3">Valor 3</option>
</select>
```



## <label></label>

El elemento label no es un componente interactivo, pero es muy importante su utilización con ellos, ya que asigna una etiqueta o rótulo (no confundir con etiqueta html) enlazada al componente, y se renderiza.

Este elemento debe tener el atributo **for**, que utiliza el **id** elegido para el componente interactivo:

```
<label for="nombre-apellido">Nombre y apellido</label>
<input type="text" id="nombre-apellido" />
```

Nombre y apellido

## Atributos de validación

Todos los elementos interactivos o de introducción de datos del formulario constan de varios atributos que podemos utilizar para restringir los datos que el usuario introduce. Solo citaremos los genéricos. Y como siempre, debemos ver la documentación.

**IMPORTANTE**, si bien esto valida los datos de la interfaz, NO VALIDA el envío de estos datos, ya que se puede saltar esta validación con conocimientos básicos de JavaScript. La validación de seguridad propiamente dicha se realiza del lado del servidor siempre.

### required

Si está presente, establece que el campo no debe estar vacío al momento de ser enviado el formulario. Se indica con el atributo **required**, cuyo valor por defecto es true.

```
<input type="text" id="nombre-registro" name="nombre" required />
```

### disabled

Si está presente, el campo es deshabilitado y el usuario no puede ingresar valores. Además, es ignorado al momento del envío del formulario. Se indica con el atributo **disabled**, cuyo valor por defecto es true.

```
<input type="text" id="nombre-registro" name="nombre" disabled />
```

Es muy utilizado conjuntamente con JavaScript habilitando y deshabilitando campos según necesidad.

### readonly

Si está presente, indica que el valor es de solo lectura y no podrá ser modificado por el usuario. Su valor es parte del envío del formulario. Se indica con **readonly**, cuyo valor por defecto es true.

```
<input type="text" id="nombre-registro" name="nombre" readonly />
```

Al igual que **disabled**, **readonly** es muy utilizado conjuntamente con JavaScript.

## Multimedia y recursos externos

El documento HTML también soporta nativamente otros medios como imágenes, videos, audio, dibujo, 3D, entre otros medios. Vamos a repasar algunos, pero para mayor información, deberán recurrir a la documentación.

### <img />

Es el elemento encargado de vincular un recurso de tipo imagen. Este elemento no es nuevo, pero sí fue mejorando con nuevos atributos.

Su estructura básica es:

```

```

El atributo **src** es el encargado de vincular con la imagen. Hay que verificar si el formato de imagen es compatible.

### <video></video>

Este elemento se encarga de incluir el recurso video. En su estructura más básica se indica el recurso recurso con el atributo **src**

```
<video src="video.mp4"></video>
```

Como existen varios formatos de video en el mercado, podemos anidar `<source />` que permite indicar al Navegador varios formatos en donde éste eligirá el compatible. Esto se debe a que cada Navegador Web puede o no soportar el formato de archivo video.

```
<video controls>
  <source src="video.webm" type="video/webm" />
  <source src="video.mp4" type="video/mp4">
</video>
```

Aquí indicamos dos formatos de video, **webm** y **mp4**. Además se agrega **type**, otro atributo que ayuda al Navegador a entender el tipo de recurso. Estos tipos de recursos

se especifican como **MIME types** o **Media Types**. Podemos encontrarlos también como **Content-Type**

Para ampliar los formatos, recurrir a la documentación:

[Media type and format guide: image, audio, and video content - Web media technologies | MDN](#)

### **MIME Types**

Los **MIME Types** son especificaciones de La Web. Podemos encontrar una lista aquí [Media Types](#)

También, podemos indicar con el atributo **controls** si debe o no mostrar los controles de reproducción. Como está indicado **controls** en el ejemplo, equivale a indicar **controls="true"**

Ver más en:

[video - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

<audio></audio>

Al igual que el elemento video, audio sigue el mismo formato:

```
<audio src="audio.mp3"></audio>
```

También podemos indicar distintos tipos con <source...

```
<audio controls autoplay="true">
  <source src="audio.mp3" type="audio/mp3" />
  <source src="audio.ogg" type="audio/ogg" />
</audio>
```

Vemos aquí el atributo **autoplay**, que indica que se debe reproducir automáticamente. Hoy en día los Navegadores no permiten inicializar de esta manera, solo cuando un usuario interactúe con el elemento.

Ver más en:

[Audio - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

### <iframe></iframe>

Este elemento permite incrustar otro documento html. Funciona como un marco o ventana. Al trabajar con este elemento debemos conocer las políticas de seguridad y cómo debemos utilizarlo de manera segura, ya que al introducir una página externa, se puede tener control sobre la página que lo contiene.

```
<iframe src="https://dominio.com/pagina.html"></iframe>
```

Es muy utilizado para incrustar servicios externos como mapas (google maps, openstreet maps), o videos (youtube, vimeo).

Ver más en:

[<iframe>: el elemento Inline Frame - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

### <object></object>

Este elemento puede contener cualquier tipo de recurso. Debemos explicitar el atributo **data**, que es el URL del recurso y el atributo **type**, que es el tipo de recurso (Mime Type o Media Type) cuya especificación vimos en el apartado del elemento <video>

```
<object data="documento.pdf" type="application/pdf"></object>
```

### Elija la opción correcta

1. Un documento HTML se compone de cuatro elementos fundamentales:
  - a) el DOCTYPE html, el html, el head y el body
  - b) el DOCTYPE html, el header, el body y el footer
  - c) el DOCTYPE html, el html, el body y el footer
2. Los elementos a renderizar dentro de un documento HTML se deben colocar en:
  - a) el DOCTYPE html
  - b) el body
  - c) el head
3. El marcado semántico:
  - a) permite dar sentido a los contenidos, mejorar el orden y mejorar la indexación de los buscadores.
  - b) permite dar formato a los contenidos, mejorar la accesibilidad y mejorar la indexación de los buscadores.
  - c) permite dar sentido a los contenidos, mejorar la accesibilidad y mejorar la indexación de los buscadores.
4. El elemento ancla <a href="una\_url">Descripción</a>:
  - a) es un elemento que permite remarcar un texto
  - b) es un elemento que permite vincular recursos
  - c) es un elemento que permite formatear un contenido

5. A un elemento <form> </form> se le deben asignar dos atributos obligatorios para enviar contenidos al servidor:
- a) el atributo post y el atributo method
  - b) el atributo action y el atributo method
  - c) el atributo action y el atributo get
6. Los elementos HTML anidados dentro de un elemento <form> </form>, para poder ser identificados en una petición HTTP, deben tener obligatoriamente el atributo:
- a) name
  - b) id
  - c) alt
7. Para que un elemento anidado en <form> </form> sea marcado como obligatorio de completar por parte del usuario, debemos asignar el atributo:
- a) needed
  - b) completed
  - c) required
8. Los elementos multimedia permiten:
- a) incluir archivos multiformato externos dentro de un documento HTML
  - b) vincular archivos multiformato externos al que podemos visitar
  - c) vincular archivos multiformato internos fuera de un documento HTML

### Opciones correctas<sup>33</sup>

---

<sup>33</sup> 1.a) el DOCTYPE html, el html, el head y el body. 2. b) el body. 3. c) permite dar sentido a los contenidos, mejorar la accesibilidad y mejorar la indexación de los buscadores. 4. b) es un elemento que permite vincular recursos. 5. b) el atributo action y el atributo method. 6. a) name. 7. c) required. 8. a) incluir archivos multiformato externos dentro de un documento HTML.

## SP2/H4: Accesibilidad Web

La accesibilidad web es la práctica que permite que todo usuario con algún tipo de discapacidad pueda acceder a la web. Este es un tema muy extenso e importante del cual debemos ser conscientes a la hora del desarrollo, ya que no es frecuente encontrar buenas aplicaciones web que sigan estos estándares.

Las Directrices de Accesibilidad para el Contenido Web o WCAG, por sus siglas en inglés, son algunas de las directrices incorporadas por la WAI<sup>34</sup> (Iniciativa de Accesibilidad Web) que es parte del World Wide Web Consortium (W3C), encargada de los estándares web.

### Los cuatro principios de la WCAG

Citamos aquí los cuatro principios<sup>35</sup> sobre los cuales se fundamenta la accesibilidad web con sus pautas, de manera que podamos analizar críticamente la construcción de nuestras aplicaciones web:

**1. Perceptible:** la información y los componentes de la interfaz de usuario deben ser mostrados a los usuarios en formas que ellos puedan entender.

- **Pauta 1.1 Texto alternativo:** Proporciona texto alternativo para el contenido que no sea textual, así podrá ser transformado en otros formatos que la gente necesite, como caracteres grandes, lenguaje braille, lenguaje oral, símbolos o lenguaje más simple.
- **Pauta 1.2 Contenido multimedia dependiente del tiempo:** Proporcione alternativas sincronizadas para contenidos multimedia sincronizados dependientes del tiempo.
- **Pauta 1.3 Adaptable:** Crear contenido que pueda ser presentado de diferentes formas sin perder ni información ni estructura.

---

<sup>34</sup> [WAI Interest Group \(WAI IG\) | Web Accessibility Initiative \(WAI\) | W3C](#)

<sup>35</sup> [Introduction to Understanding WCAG | WAI | W3C](#)

- Pauta 1.4 **Distinguible**: Facilitar a los usuarios ver y escuchar el contenido incluyendo la distinción entre lo más y menos importante.

**2. Operable:** Los componentes de la interfaz de usuario y la navegación debe ser manejable.

- Pauta 2.1 **Teclado accesible**: Poder controlar todas las funciones desde el teclado.
- Pauta 2.2 **Tiempo suficiente**: Proporciona tiempo suficiente a los usuarios para leer y utilizar el contenido.
- Pauta 2.3 **Ataques epilépticos**: No diseñar contenido que pueda causar ataques epilépticos.
- Pauta 2.4 **Navegación**: Proporciona formas para ayudar a los usuarios a navegar, a buscar contenido y a determinar dónde están estos.
- Pauta 2.5 **Modalidades de entrada**: Facilitar a los usuarios operar la funcionalidad a través de varios métodos de entrada además del teclado.

**3. Comprensible:** La información y las operaciones de los usuarios deben ser comprensibles.

- Pauta 3.1 **Legible**: Hacer contenido de texto legible y comprensible.
- Pauta 3.2 **Previsible**: Hacer la apariencia y la forma de utilizar las páginas web previsibles.
- Pauta 3.3 **Asistencia a la entrada de datos**: los usuarios de ayuda evitarán y corregirán errores.

**4. Robustez:** el contenido debe ser suficientemente robusto para que pueda ser bien interpretado por una gran variedad de agentes de usuario, incluyendo tecnologías de asistencia.

- Pauta 4.1 **Compatible**: Maximiza la compatibilidad con los agentes de usuario actuales y futuros, incluyendo tecnologías de asistencia.

## ARIA

ARIA es el acrónimo de Aplicaciones Ricas de Internet Accesibles (Accessible Rich Internet Applications)<sup>36</sup>, que son una serie de atributos para los elementos HTML que permiten mejorar la accesibilidad a los contenidos de una página, normalmente para los lectores o readers de pantalla.

Tengamos en cuenta que muchos de estos atributos se crearon antes del etiquetado semántico que se incluyó a partir de la versión 5 de HTML, entonces, la primera regla del ARIA es utilizar siempre las etiquetas semánticas y los atributos nativos del HTML 5, y luego utilizar los atributos de rol o role en inglés.

### role

El atributo **role** da información sobre la función del elemento HTML:

```
<button type="submit">Enviar formulario</button>
```

En el ejemplo del botón, elemento HTML **<button>**, es evidente su sentido, además especificamos el atributo **type**. Tanto la etiqueta como el atributo son nativos de HTML, y no quedan dudas de su significado.

Muy distinto sería lo siguiente:

```
<div role="alert">Este es un mensaje para el usuario</div>
```

En el ejemplo de arriba, el elemento HTML **div** es un elemento genérico y no dice mucho, por lo tanto utilizaremos el atributo **role**.

En la documentación de WAI encontraremos el listado de roles<sup>37</sup>.

### aria-\*

También encontramos los atributos **aria-\*** (el asterisco indica que debe llevar un nombre de alguna propiedad, por ejemplo, **aria-valuemin**). Estos atributos permiten

<sup>36</sup> <https://developer.mozilla.org/es/docs/Web/Accessibility/ARIA>

<sup>37</sup> [Accessible Rich Internet Applications \(WAI-ARIA\) 1.1](#)

brindar más información de manera dinámica combinado con la utilización de JavaScript para cambiar sus valores.

Existe una lista extensa<sup>38</sup> de estos atributos que pueden utilizar.

*Algo importante a destacar es que el tema de la accesibilidad llevado a la práctica puede ser contraproducente si no se utiliza bien, por lo que debemos consultar siempre la documentación para un correcto uso.*

---

<sup>38</sup> [Estados y propiedades de ARIA - Accesibilidad | MDN](#)

### Elija la opción correcta

1. ¿Cuándo debemos utilizar los cuatro principios del WCAG en el desarrollo de aplicaciones web?
  - a) Siempre
  - b) Cuando los usuarios de la aplicación sean personas con algún tipo de discapacidad
  - c) Nunca
2. Los cuatro principios de la WCAG son:
  - a) Perceptible, Modifiable, Comprensible, Robustez
  - b) Perceptible, Operable, Comprensible, Robustez
  - c) Perceptible, Operable, Visible, Robustez
3. ¿Qué es ARIA en el desarrollo web?
  - a) Una organización de accesibilidad web.
  - b) Acrónimo de la Iniciativa de Accesibilidad Web.
  - c) Atributos para mejorar la accesibilidad en aplicaciones web.
4. ¿Cuál es la primera regla en el uso de ARIA?
  - a) Utilizar siempre los atributos de ARIA antes que las etiquetas semánticas.
  - b) Utilizar siempre las etiquetas semánticas y atributos nativos de HTML 5 en primer lugar.
  - c) Usar el atributo role en lugar de etiquetas HTML.

5. ¿Para qué se utiliza el atributo "role" en HTML?

- a) Para definir el tipo de contenido de un elemento HTML.
- b) Para indicar un identificador en un elemento HTML.
- c) Para marcar un elemento HTML como no accesible.

**Opciones correctas<sup>39</sup>**

---

<sup>39</sup> 1. a) Siempre. 2. b) Perceptible, Operable, Comprensible, Robustez 3. c) Atributos para mejorar la accesibilidad en aplicaciones web. 4. b) Utilizar siempre las etiquetas semánticas y atributos nativos de HTML 5 en primer lugar. 5. a) Para definir el tipo de contenido de un elemento HTML.

## SP2/H5: Buenas prácticas

Siempre es necesario seguir buenas prácticas a la hora de organizar el código HTML, sobre todo para la lectura y mantenimiento. Tanto si trabajamos en soledad o en equipo, la buena organización permite una lectura rápida de lo que se hizo.

En primer lugar, diremos que es importante no envolver elementos HTML porque sí, es preferible siempre la utilización de la menor cantidad posible de código HTML, sobre todo cuando estamos anidando un elemento dentro de otro elemento HTML.

Siguiendo con la anidación, repetimos algo que ya dijimos anteriormente, no anidar elementos de bloque en elementos de línea.

También podemos aplicar una buena organización de los atributos de un elemento HTML:

1. class
2. id, name
3. data-\*
4. src, for, type, href, value
5. title, alt
6. role, aria-\*
7. tabindex
8. style

```
<p class="" id="" role="">  
    Texo de un párrafo  
</p>          1   2   3   4  
<input type="text" class="" id="" name="" title="">
```

Y recordemos siempre de implementar protocolos de comunicación con el equipo. También tener la costumbre, aunque trabajemos en soledad, de organizar los

materiales, incluir comentarios, utilizar repositorios y control de versiones. Todo esto lo vamos a agradecer cuando debamos actualizar un proyecto viejo y no se vuelva un dolor de cabeza.



Link del video: <https://www.youtube.com/watch?v=3ubN-zGeMKI>

### Elija la opción correcta

1. ¿Cuándo debemos seguir buenas prácticas a la hora de trabajar?
  - a) Siempre que no estemos trabajando en equipo.
  - b) Siempre.
  - c) Solo cuando trabajamos en equipo.
  
2. ¿Por qué es preferible utilizar la menor cantidad posible de código HTML y evitar la anidación excesiva?
  - a) Para facilitar la lectura y el funcionamiento del código.
  - b) Para facilitar la lectura y el mantenimiento del código.
  - c) Para facilitar el funcionamiento y el mantenimiento del código.
  
3. ¿Cuándo podemos anidar elementos HTML de bloque dentro de elementos HTML de línea?
  - a) Nunca
  - b) Sólo cuando sea necesario
  - c) Siempre que podamos
  
4. ¿En qué situaciones es recomendable utilizar comentarios en el código?
  - a) Cuando hay que marcar una sección para realizar un cambio más tarde.
  - b) Solo si trabajamos en equipo.
  - c) Siempre que podamos.

5. La utilización de repositorios y control de versiones es necesaria cuando:

- a) estamos en los inicios de un proyecto.
- b) en todas las etapas de un proyecto.
- c) cuando se finaliza el proyecto.

**Opciones correctas<sup>40</sup>**

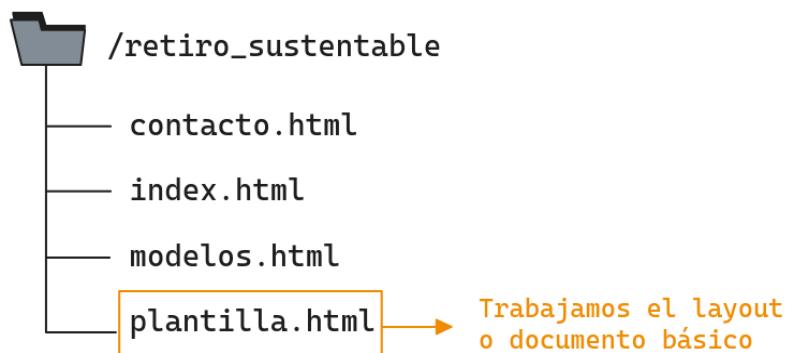
---

<sup>40</sup> 1.b) Siempre. 2.b) Para facilitar la lectura y el mantenimiento del código. 3. a) Nunca. 4. c) Siempre que podamos. 5.b) en todas las etapas del proyecto.

### Creación de la estructura HTML

Para el desarrollo de la estructura HTML nos apoyamos en el diseño: bocetos, wireframe y mockup.

En primer lugar, creamos la estructura básica del proyecto:



Como primera medida identificamos la estructura básica que es la estructura que se repetirá en todos los documentos HTML con sus respectivos bloques principales de renderizado, y creamos un documento HTML "*plantilla.html*"

#### Bloque head:

Se plantean todos los elementos necesarios tanto para los recursos, así como como los meta elementos. Los más importantes son:

- <meta charset="UTF-8" />  
Juego de caracteres
- <meta name="viewport" content="width=device-width, initial-scale=1.0" /><sup>41</sup>  
Optimiza el “viewport” ( lo que se ve de la ventana) para la correcta visualización en dispositivos móviles.

<sup>41</sup> [Viewport meta tag - HTML: HyperText Markup Language | MDN](#)

- **<meta name="description" content="" />**  
Crea una descripción de los contenidos del documento HTML. Importante para el SEO.
- **<link rel="shortcut icon" href="" type="image/svg+xml" />**  
Enlaza un ícono para la pestaña de la página.
- **<title>Copan | Retiro Sustentable</title>**  
Título del documento HTML. Figura en la pestaña al igual que el ícono anterior.

## Bloques principales de renderizado en <body>:

- **<header> (Cabecera)**  
En donde incluiremos la imagen que representa la marca o nombre del proyecto así como el menú principal de navegación.
- **<main> (Principal)**  
En este bloque irán todos los contenidos particulares de cada vista o página.
- **<footer> (Pie)**  
Bloque con información genérica, links y datos de contacto.

```

<!--
    Estructura base de la aplicación
-->
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8" />
        <meta name="viewport" content="width=device-width, initial-scale=1.0" />
        <!-- Esta meta, entre otras, son importantes para el SEO -->
        <meta name="description" content="">
        <!-- Icono que se mostrará en la pestaña al lado del título
            (en este caso imagen SVG) -->
        <link
            rel="shortcut icon"
            href=""
            type="image/svg+xml"
        />
        <!-- recursos css -->
        <!-- recursos script -->

        <!-- título que se mostrará en la pestaña -->
        <title>Copan | Retiro Sustentable</title>
    </head>
    <body>
        <!-- cabecera -->
        <header>
            <!-- contenidos: -->
            <!-- - imagotipo, logotipo o título -->
            <!-- - menu -->
        </header>
        <!-- main -->
        <main>
            <!-- contenido de cada página -->
        </main>
        <!-- pie -->
        <footer>
            <!-- contenidos: link redes sociales y contacto -->
        </footer>
    </body>
</html>

```

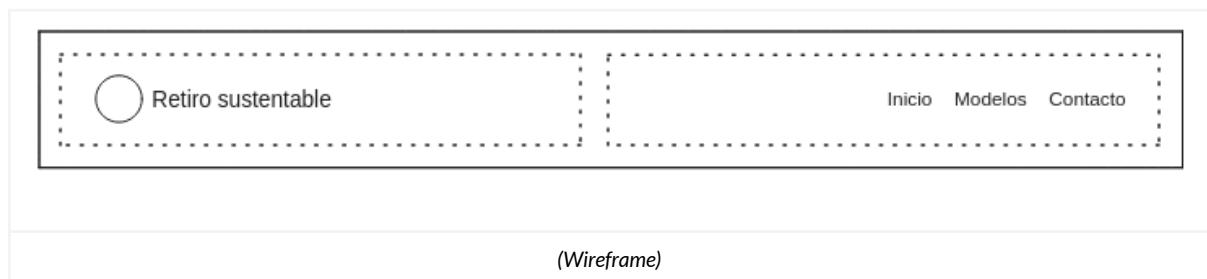
A continuación se estructuran los distintos componentes de las zonas incluídas en el **<header>** (cabecera) y el **<footer>** (pie) que son los que se repetirán en todas las vistas.

Comenzamos a pensar la estructura desde afuera hacia adentro e ir organizando los elementos HTML que se van anidando como cajas.

**IMPORTANTE:** Si bien tenemos en mente el diseño con sus respectivos “alineados” y tamaños, debemos tener en cuenta que al no tener estilos todavía, no lo vamos a ver renderizado de la manera en que se presentan las imágenes.

## Estructura <header>

Se plantea una cabecera con dos cajas anidadas, una que contiene el imagotipo y la segunda el menú principal.

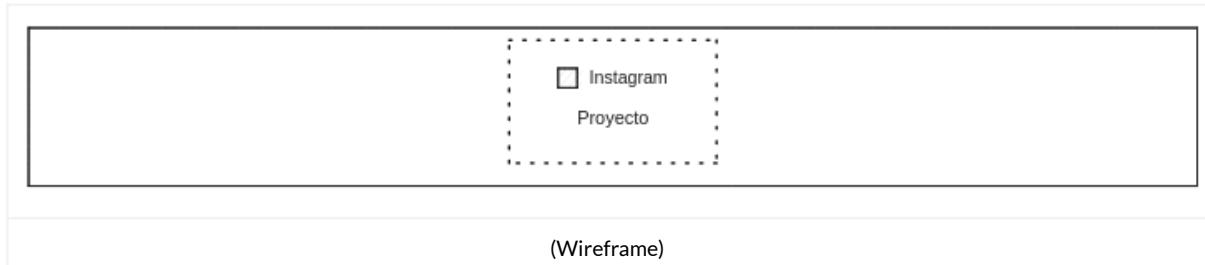


Dentro de las cajas anidadas se distribuyen más elementos HTML. De esta manera vamos armando la estructura.

```
<!-- cabecera -->
<header>
    <div>
        <!-- link con imagen de logo empresa -->
        <a href="#">
            <img src="" alt="" />
        </a>
    </div>
    <!-- menu principal -->
    <nav>
        <ul>
            <li><a href="index.html">Inicio</a></li>
            <li><a href="modelos.html">Modelos</a></li>
            <li><a href="contacto.html">Contacto</a></li>
        </ul>
    </nav>
</header>
```

## Estructura <footer>

Se plantea el pie con una lista desordenada para los links y datos.



```
<!-- pie -->
<footer>
  <ul>
    <!-- link redes -->
    <li>
      <a href="#"><img src="" alt="" /></a>
    </li>
    <!-- teléfono -->
    <li>Centro y showroom | +54 9 111111</li>
    <!-- Logo empresa -->
    <li>
      Desarrollado por
      <img src="" alt="" />
    </li>
  </ul>
</footer>
```

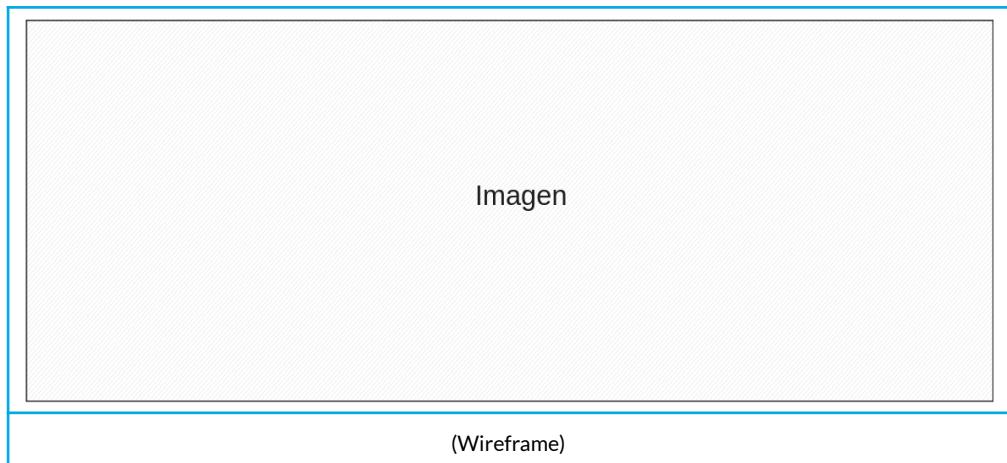
## Contenidos de las tres pantallas

Ahora, con la plantilla realizada, la utilizaremos en las tres páginas, y comenzamos a delinear los contenidos.

### Página de Inicio

Para la página de inicio se determinan tres bloques.

La imagen:



```
<figure>
    <img src="" alt="" />
</figure>
```

## La sección de descripción del proyecto:

### Proyecto

---

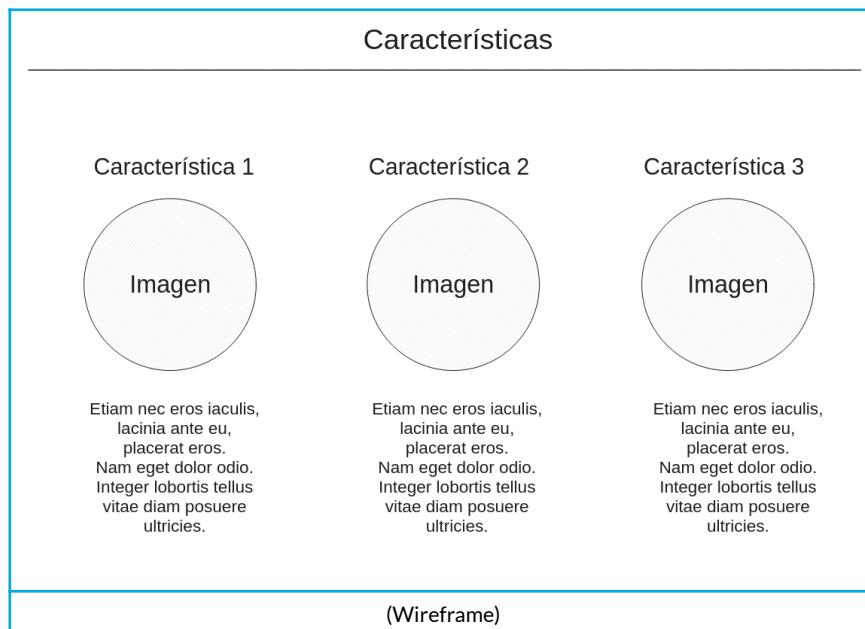
Etiam nec eros iaculis, lacinia ante eu, placerat eros.  
Nam eget dolor odio. Integer lobortis tellus vitae diam posuere ultricies.

Etiam nec eros iaculis, lacinia ante eu, placerat eros.  
Nam eget dolor odio. Integer lobortis tellus vitae diam posuere ultricies.

(Wireframe)

```
<section>
    <h1>Proyecto</h1>
    <p>
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit. Incidunt,
        voluptatibus!
    </p>
    <p>
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit. Incidunt,
        voluptatibus!
    </p>
</section>
```

## La sección de las características:



```
<section>
  <h2>Características</h2>
  <div>
    <article>
      <h3>Característica 1</h3>
      <figure>
        <img src="" alt="" />
      </figure>
      <p>
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit.
        Incidunt, voluptatibus!
      </p>
    </article>
    <article>
      <h3>Característica 2</h3>
      <figure>
        <img src="" alt="" />
      </figure>
      <p>
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit.
        Incidunt, voluptatibus!
      </p>
    </article>
    <article>
      <h3>Característica 3</h3>
      <figure>
        <img src="" alt="" />
      </figure>
      <p>
        Lorem ipsum dolor sit amet, consectetur
        adipisicing elit.
        Incidunt, voluptatibus!
      </p>
    </article>
  </div>
</section>
```

## Página de Modelos

La página **modelos.html** es donde tenemos planificado un enfoque SPA a fin de que la interactividad sea dinámica.

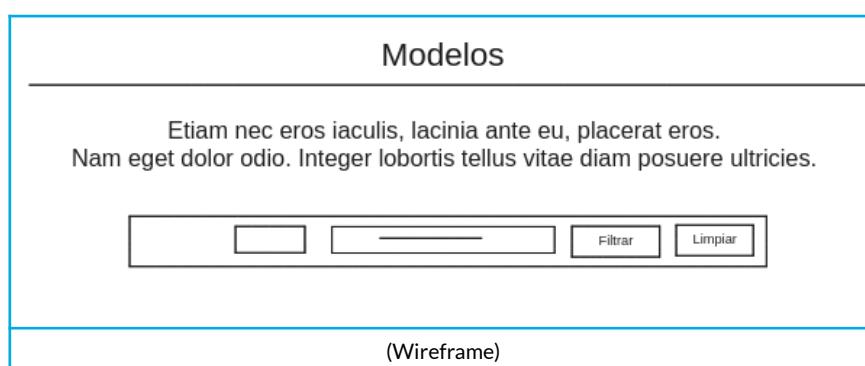
La estructura general sería:

```
<main>
    <h1>Modelos</h1>
    <p>
        Lorem, ipsum dolor sit amet consectetur adipisicing elit. Ipsa
        asperiores id, exercitationem quod totam preferendis nam a fugiat
        tempore voluptatum?
    </p>
    <form id="id-modelos-filtro">
        <!-- filtro -->
    </form>
    <div id="modelos-app" aria-live="polite">
        <!-- Lista de modelos -->
    </div>
</main>
```

← Sección donde se injectarán los contenidos HTML dinámicos

Vemos arriba que ya indicamos los atributos id para la manipulación que haremos luego con JavaScript. También, indicamos en el área de injectado de HTML, un atributo **aria-live**<sup>42</sup> a fin de notificar que es un elemento que se actualizará.

Completemos ahora el bloque de filtro:



<sup>42</sup> <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Attributes/aria-live>

En el bloque de filtro debemos indicar todos los elementos necesarios así como sus atributos.

### 1 - Filtrado por cantidad de dormitorios:

```
<label for="id-dormitorios">  
    Dormitorios  
    <select name="dormitorios" id="id-dormitorios">  
        <option value="0">todos</option>  
        <option value="1">1</option>  
        <option value="2">2</option>  
        <option value="3">3</option>  
        <option value="4">4</option>  
    </select>  
</label>
```

2 - Filtrado por metros cuadrados. Aquí vamos a considerar dos elementos HTML, el elemento `<input>` del tipo “range”<sup>43</sup>, y el tipo `<output>`<sup>44</sup>. Ambos elementos funcionan juntos para seleccionar y mostrar los metros cuadrados.

```
<input  
    id="id-rango-m2"  
    type="range"  
    step="5"  
    min="40"  
    max="100"  
    value="100"  
/>
```

```
<output  
    id="id-max-m2"  
    for="id-rango-m2" ← Conecta con  
    form="id-modelos-filtro"  
    >  
    100  
</output>
```

(Observemos el atributo `for` que conecta con el `<input type="range">`)

<sup>43</sup> [<input type="range"> - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

<sup>44</sup> [The Output element - HTML: HyperText Markup Language | MDN](#)

Ahora veamos el “range” completo:

```
<label for="id-rango-m2">
    max
    <input
        name="m2"
        id="id-rango-m2"
        type="range"
        step="5"
        min="40"
        max="100"
        value="100"
    />
    <output
        id="id-max-m2"
        for="id-rango-m2"
        form="id-modelos-filtro">
        100
    </output>M2
</label>
```

Esta primera parte quedaría de la siguiente manera:

```
<h1>Modelos</h1>
<p>
    Lorem, ipsum dolor sit amet consectetur adipisicing elit. Ipsa
    asperiores id, exercitationem quod totam preferendis nam a fugiat
    tempore voluptatum?
</p>
<form id="id-modelos-filtro">
    <label for="id-dormitorios">
        Dormitorios
        <select name="dormitorios" id="id-dormitorios">
            <option value="0">todos</option>
            <option value="1">1</option>
            <option value="2">2</option>
            <option value="3">3</option>
            <option value="4">4</option>
        </select>
    </label>
    <label for="id-rango-m2">
        max
        <input
            name="m2"
            id="id-rango-m2"
            type="range"
            step="5"
            min="40"
            max="100"
            value="100"
        />
        <output
            id="id-max-m2"
            for="id-rango-m2"
            form="id-modelos-filtro">
            100
        </output>M2
    </label>
    <button id="id-filtrar"><img src="" /> Filtrar</button>
    <button id="id-limpiar" type="reset">Limpiar</button>
</form>
```

El renderizado de la primera parte quedaría así:

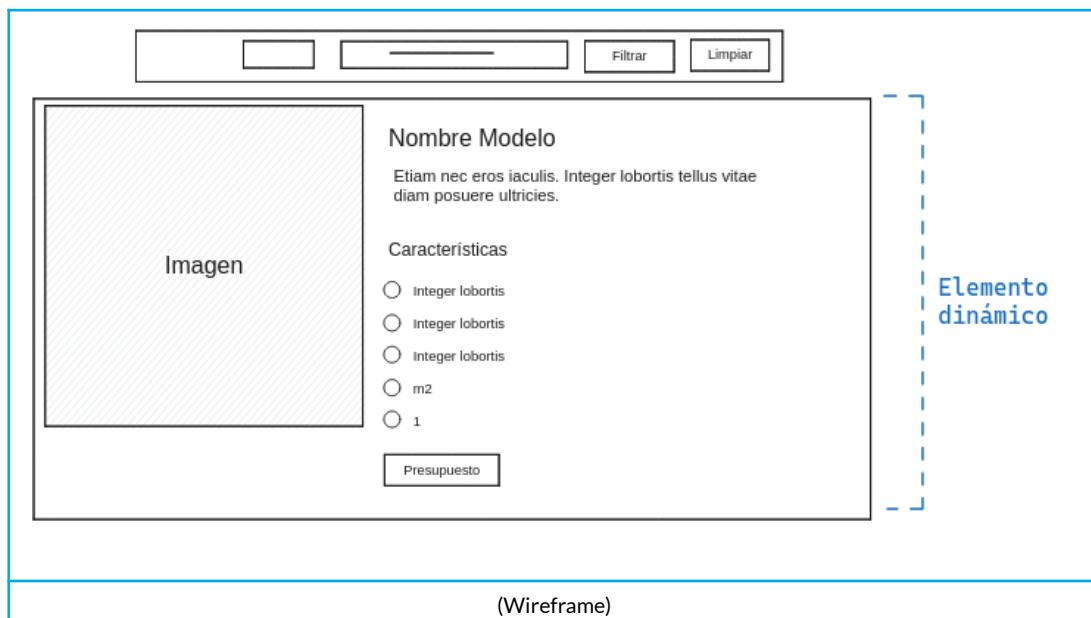
# Modelos

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Ipsa asperiores id, exercitationem quod totam preferendis nam a fugiat tempore voluptatum?

Dormitorios  max  100 M2

Continuamos con los modelos. En esta parte debemos detectar cuál es el elemento que responderá con la acción de filtrado.

Si nos fijamos bien, será el modelo con su imagen y características:



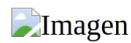
Utilizaremos el elemento <article> a fin de que sea semántico. Además, prestemos atención al uso de atributos ARIA en el caso de imágenes decorativas como los íconos:

```
<main>
    <!-- ... Contenido HTML -->

    <div id="modelos-app" aria-live="polite">
        <!-- modelo -->
        <article>
            <figure>
                <!-- imagen modelo -->
                <img src="" alt="" />
            </figure>
            <div>
                <h2>Nombre modelo</h2>
                <p>
                    Lorem ipsum dolor, sit amet consectetur
                    adipisicing elit. Nam, iusto.
                </p>
                <h3>Características</h3>
                <ul>
                    <li>
                        <img src="" alt="" aria-hidden="true" />
                        Característica 1
                    </li>
                    <li>
                        <img src="" alt="" aria-hidden="true" />
                        Característica 2
                    </li>
                    <li>
                        <img src="" alt="" aria-hidden="true" />
                        Característica 3
                    </li>
                    <li>
                        <img src="" alt="" aria-hidden="true" />
                        Característica 4
                    </li>
                    <li>
                        <img src="" alt="" aria-hidden="true" />
                        Característica 5
                    </li>
                </ul>
                <button class="">Presupuesto</button>
            </div>
        </article>
    </div>
</main>
```

Se repetirán los artículos con la misma estructura HTML

Y su renderizado:



## Nombre modelo

Lorem ipsum dolor, sit amet consectetur adipisicing elit. Nam, iusto.

### Características

- ImagenCaracterística 1
- ImagenCaracterística 2
- ImagenCaracterística 3
- ImagenCaracterística 4
- ImagenCaracterística 5

Presupuesto

## Ventana emergente o popup

Dejaremos listo el código del formulario de presupuesto también. Este código será manejado luego con JavaScript, y es parte del enfoque SPA en la página **modelos.html**:

The diagram illustrates a modal window titled "Modelo". The window has a close button ("X") at the top right. Inside, there are two checkboxes labeled "Ampliar Característica 2" and "Ampliar Característica 2". Below these are input fields for "Nombre" and "Email". There is also a larger text area for "Comentarios". At the bottom is a greyed-out button labeled "Presupuestar".

Se utiliza el elemento <**dialog**><sup>45</sup>. Vamos a verificar que todos los atributos estén correctamente utilizados.

```

<dialog>
  <div>
    <button id="btn-cerrar-popup">
      <img src="" alt="Cerrar popup" />
    </button>
  </div>
  <div>
    <div>
      <h3>Nombre modelo</h3>
      <p>Enviar datos para presupuesto</p>
      <form>
        <h5>Agregar características</h5>
        <ul>
          <li>
            <input id="id-caract-1" name="caract-1" type="checkbox" value="1" />
            <label for="id-caract-1">Ampliar Característica 1</label>
          </li>
          <li>
            <input id="id-caract-1" name="caract-1" type="checkbox" value="1" />
            <label for="id-caract-1">Ampliar Característica 2</label>
          </li>
        </ul>
        <label for="id-nombre"> Nombre </label>
        <input id="id-nombre" name="nombre" type="text" minlength="3" required />
        <label for="id-email"> Email </label>
        <input id="id-email" name="email" type="email" required />
        <label for="id-email"> Comentarios </label>
        <textarea name="comentario" id="id-comentario" cols="30" rows="10" minlength="4" required></textarea>
        <button type="submit">Presupuestar</button>
      </form>
    </div>
  </div>
</dialog>

```

---

<sup>45</sup> [<dialog> - HTML: Lenguaje de etiquetas de hipertexto | MDN](#)

## Página de Contacto

En la página contacto solo irá un formulario con una descripción general.

Debemos prestar especial cuidado a la hora de crear el formulario a fin de tener todos los atributos necesarios para el envío de datos y la validación del mismo:

```
<main>
  <h1>Contacto</h1>
  <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Enim, unde!</p>
  <form id="id-form-contacto" action="" method=""> ← Escript que recibe
    <label for="id-nombre">Nombre</label>           los datos y el método
    <input
      type="text"
      name="nombre" ← Envío de datos
      id="id-nombre"
      minlength="3" ← Validación
      required
    />                                     ] Por ejemplo en el campo nombre
    <label for="id-email">Email</label>
    <input type="email" name="email" id="id-email" required />
    <label for="id-mensaje">Mensaje</label>
    <textarea
      name="mensaje"
      id="id-mensaje"
      cols="30"
      rows="10"
      minlength="3"
      required
    ></textarea>
    <div>
      <button class="boton-claro" type="reset">Limpiar</button>
      <button class="boton-oscuro" type="submit">Enviar</button>
    </div>
  </form>
</main>
```

## SP2/Ejercicio por resolver

### Etapa 2: Creación de la estructura HTML

Para el proyecto de LA CAFETERÍA, se seguirán los mismos lineamientos generales que el proyecto de *Copan Arquitectura*.

Se crearán todas páginas HTML además de las secciones de la SPA que serán renderizadas en el front-end (CSR).

Para el desarrollo se deberá tener en cuenta el diseño estipulado a fin de estructurar correctamente el HTML. Como sabemos, la codificación no será definitiva, pero debemos respetar el diseño planteado:

1. Detectar la estructura base que se repetirá en todas las vistas y estructurar un layout común para todas las páginas.
2. Desglosar la interfaz por componentes funcionales mínimos: Encabezado, menú de navegación, pie de página, componentes de información compuestos (imagen, título, texto).
3. Determinar los elementos semánticos a utilizar.
4. Crear los meta elementos necesarios para un buen SEO para cada página.

## SP2/Evaluación de paso

### Elija la opción correcta

1. ¿Qué tipo de herramientas y organización son importantes para el desarrollo Front-end?
  - a) Navegador Web, IDE, y un plan de trabajo.
  - b) Navegador Web, Servidor, y un plan de trabajo.
  - c) Base de datos, Control de versiones, y un plan de trabajo.
2. ¿Cuál es el propósito del entorno de desarrollo integrado (IDE)?
  - a) Facilitar la organización de archivos en tu computadora.
  - b) Ayudar a trabajar de manera más eficiente proporcionando diversas herramientas.
  - c) Actuar como un servidor web para pruebas en tiempo real.
3. ¿Qué es un repositorio en el contexto del control de versiones?
  - a) Un lugar físico que almacena distintas versiones de tu código.
  - b) Un navegador web para ver las aplicaciones en línea.
  - c) Una herramienta para crear gráficos en tus proyectos.
4. ¿Qué es el elemento HTML en el contexto del desarrollo web?
  - a) Una etiqueta HTML con sus atributos y contenidos.
  - b) Un espacio HTML para escribir código HTML .
  - c) Es uno de los atributos de una etiqueta HTML.
5. ¿Qué función tienen los atributos en HTML?
  - a) Hacer comentarios en el código HTML.
  - b) Organizar el proyecto en distintos archivos y directorios.
  - c) Dar más información al navegador de cómo debe representar el contenido de un elemento HTML.

6. ¿Cuál es el elemento HTML principal y raíz en un documento HTML?

- a) <head></head>
- b) <html></html>
- c) <body></body>

7. ¿Qué atributo se utiliza en la etiqueta <form> para indicar la URL a la que se enviarán los datos del formulario?

- a) **method**
- b) **type**
- c) **action**

8. ¿Qué elemento HTML se utiliza para introducir texto multilínea en un formulario?

- a) <input />
- b) <textarea></textarea>
- c) <select></select>

9. ¿Cuáles son los cuatro principios de la WCAG (Directrices de Accesibilidad para el Contenido Web) que fundamentan la accesibilidad web?

- a) Perceptible, Operable, Comprensible, Robusto
- b) Perceptible, Navegable, Comprensible, Robusto
- c) Perceptible, Interactivo, Legible, Robusto

10. ¿Qué es ARIA y para qué se utiliza en la accesibilidad web?

- a) ARIA es un tipo de etiqueta HTML.
- b) ARIA son las siglas de una organización de estándares web.
- c) ARIA son atributos para elementos HTML que mejoran la accesibilidad web, especialmente para lectores de pantalla.

### Opciones correctas<sup>46</sup>

<sup>46</sup> 1.a) Navegador Web, IDE, y un plan de trabajo. 2. b) Ayudar a trabajar de manera más eficiente proporcionando diversas herramientas. 3. a) Un lugar físico que almacena distintas versiones de tu código. 4. a) Una etiqueta HTML con sus atributos y contenidos. 5. c) Dar más información al navegador de cómo debe representar el contenido de un elemento HTML. 6. b) <html></html> 7. c) action. 8. b)

# Situación profesional 3: *Estilización de la interfaz*

## Creación de los estilos CSS

Finalizada la etapa 2, de lo solicitado por nuestro cliente, *Copan Arquitectura*, se debe comenzar con la estilización de los documentos HTML con código CSS basados en el diseño, que es la etapa 3, del proyecto.

Para la estilización, se recomienda hacer un repaso por el código HTML y, si hace falta, agregar elementos para agrupar y aplicar los estilos, por lo general agregamos elementos `<div>`. El uso de estos `<div>` u otro elemento, deben estar justificados por el diseño.

1. Creación y vinculación de la hoja de estilo principal a todas las páginas HTML.
2. Organizar el código por secciones comentadas para que facilite la lectura y desarrollo.
3. Siempre escribir el código de lo general a lo particular siguiendo la idea de estilo en cascada y teniendo en cuenta la especificidad.
4. Utilizar buenas prácticas, una metodología y nomenclatura para la codificación.

## Lenguaje CSS

La segunda tecnología básica que debemos abordar es el lenguaje CSS u Hojas de Estilo en Cascada ( Cascading Style Sheets).

En este apartado veremos los fundamentos que servirán como base para el estudio avanzado de CSS.

---

`<textarea></textarea>`. 9. a) Perceptible, Operable, Comprensible, Robusto. 10. c) ARIA son atributos para elementos HTML que mejoran la accesibilidad web, especialmente para lectores de pantalla.

## Uso de la documentación

Para la documentación de CSS utilizaremos:

- W3 Consortium - [CSS - Cascading Style Sheets home page](#)
- CSS en MDN - [CSS | MDN](#)
- Además de documentación, manuales, tablas de compatibilidad y recursos prácticos que será anexada a las clases.

Recordemos que la documentación debe estar siempre presente y ser consultada cada vez que tengamos dudas, no solo cuando estudiamos, sino también cuando desarrollamos.

## SP3/H1: Definición y Sintaxis

### Definición

A diferencia del lenguaje HTML, que se ocupa de la información y su estructuración. CSS es el lenguaje que permite dar o cambiar el estilo a un documento HTML, es decir, se ocupa de la **presentación**. Cuando hablamos de estilo, hablamos de modificar **tamaños, colores y tipografías**. Con el uso de CSS además podemos manejar **posiciones y comportamientos** de los elementos HTML, afectando el **Flujo Normal** del documento HTML.

*El Flujo Normal, es cómo el navegador ordena y muestra los elementos HTML por defecto, para que inicialmente la página web tenga un formato legible. En el caso de que no haya cambios desde los estilos o estos no existan, el Navegador tiene por defecto estilos y comportamientos iniciales (initial values). Desde CSS se puede modificar estos estilos y cambiar el Flujo Normal.*

CSS es un lenguaje muy extenso y van a encontrar que está dividido en módulos<sup>47</sup>. Cada día se van incorporando nuevas funcionalidades, es por ello que en este apunte vamos a conocer sus fundamentos y usos básicos, que luego nos permitirán entender y abordar aspectos más complejos.

### Sintaxis

En esta herramienta abordaremos la sintaxis de este lenguaje.

---

<sup>47</sup> [¿Qué es el CSS? - Aprende desarrollo web | MDN](#)