

FRONTEND I

MÓDULO I

CLASE I

Arquitectura Cliente-Servidor

Dentro del contexto de desarrollo web, esta arquitectura hace referencia a un modelo de comunicación que vincula a varios dispositivos con un servidor a través de Internet.

¿A qué llamamos cliente?

Son los dispositivos que hacen peticiones de servicios o recursos a un servidor. Pueden ser: una computadora, un teléfono celular, una tablet, una consola de videojuegos o cualquier implemento que pueda conectarse a una red. Dentro de Internet, el cliente suele acceder a estos servicios y recursos a través de un navegador web.

¿A qué llamamos servidor?

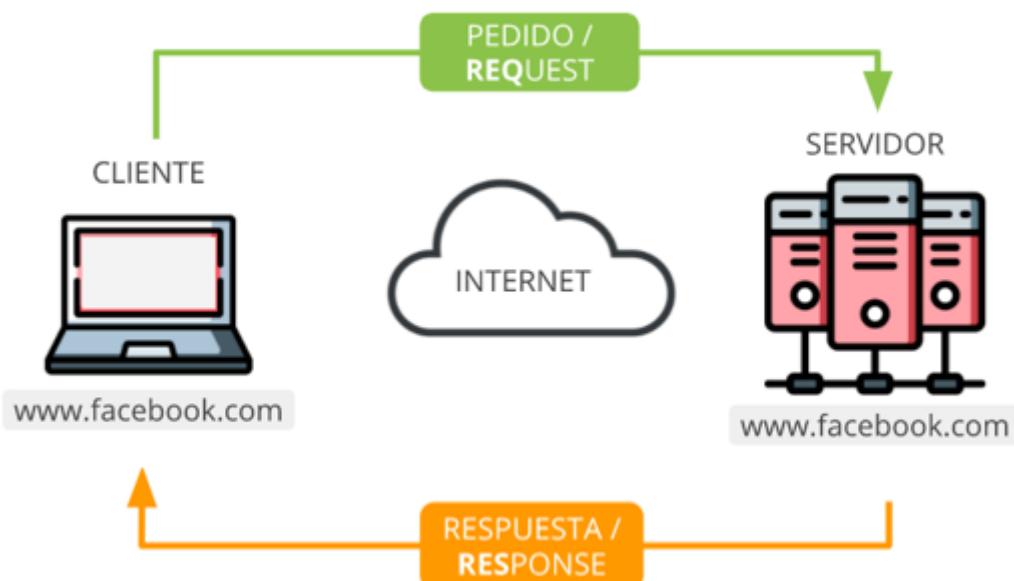
Es el equipo que brinda los servicios y recursos a los que acceden los clientes. En otras palabras, es quien responde los pedidos del cliente. Es importante tener en cuenta que la misma computadora puede ser el cliente y el servidor al mismo tiempo. De hecho, es lo más normal en el entorno de desarrollo de un sitio o aplicación web.

Los pedidos (requests)

Son las solicitudes que hacemos a través del navegador (el cliente) a un servidor. Por ejemplo, la página de Facebook que está almacenada en sus servidores.

Las respuestas (responses)

El servidor recibe nuestra solicitud, la procesa y envía como resultado una respuesta al cliente (navegador). En este ejemplo, devolverá la página principal del sitio.



¿Por qué es importante conocer este flujo request-response?

Porque dentro del mundo del desarrollo web, la mayoría de las aplicaciones tienen dos claros frentes: el front-end y el back-end.

El front-end

Es todo lo que pasa del lado del cliente (en el navegador). Aquí se incluyen todos los elementos gráficos que conforman la interfaz del sitio y también parte de su Funcionalidad. Los lenguajes que se manejan son HTML para la estructura, CSS para los estilos visuales y JavaScript para la interacción dentro del sitio.

El back-end

Es todo lo que pasa del lado del servidor. Aquí se incluye todo el funcionamiento interno y lógica del sitio. Es lo que permite que se carguen todas las peticiones solicitadas por el cliente. Podemos encontrar: bases de datos –como MySQL y MongoDB–, lenguajes, –como PHP y JavaScript para sitios webs dinámicos– y frameworks o marcos de trabajo –como Express y Laravel–.

Diferencias entre web e Internet

Ambos términos están relacionados porque uno necesita del otro para poder existir, pero son cosas diferentes.

Qué es la Web

A grandes rasgos, la Web es un sistema de distribución que puede interconectar documentos que contengan multimedia: imágenes, videos, textos y demás. Los mismos se transfieren o comunican a través de un protocolo que conocemos como HTTP (Hypertext Transfer Protocol). Estos documentos emplean el lenguaje de marcado conocido como HTML (Hypertext Markup Language) que permite realizar enlaces entre dichos archivos; lo que usualmente conocemos como links.

Qué es Internet

Podríamos decir que es la infraestructura que nos permite de alguna manera tener conectada una computadora en Argentina con otra en Japón y así cada una de las computadoras o servidores a lo largo del mundo. Este tipo de red descentralizada se utilizó en principio para fines científicos conectando universidades con instituciones estatales, como, por ejemplo, el proyecto ARPANET desarrollado por el Departamento de Defensa de los Estados Unidos en el año 1969.

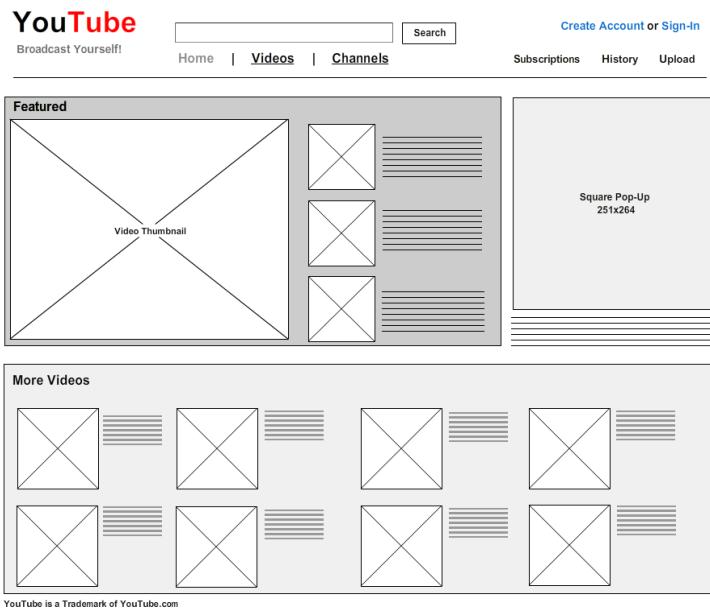
Lo importante a entender es que la Web es un modelo para compartir información que está construido sobre Internet.

Web	Internet
Sistema de distribución que interconecta documentos de hipertexto .	Red descentralizada que conecta computadoras y servidores entre sí.
Utiliza el protocolo HTTP .	Utiliza el protocolo TPC/IP , entre otros.
Información y archivos.	Infraestructura.

Al igual que en cualquier trabajo, para desarrollar nuestro proyecto web, necesitamos de herramientas para poder construirlo. En esta carrera veremos cuáles son aquellas herramientas que serán de utilidad para escribir código, para guardar las diferentes versiones que vayamos logrando y a la vez, poder trabajar de manera colaborativa, entre otras posibilidades

1.Wireframes

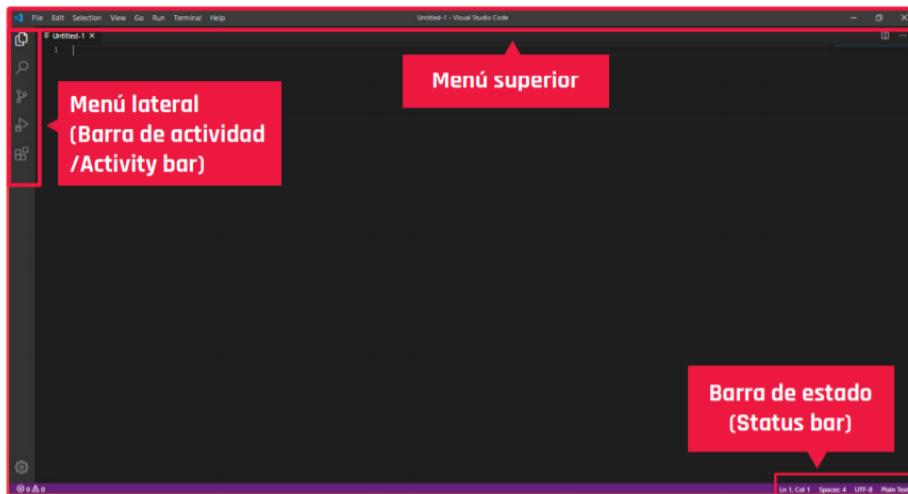
Los wireframes y los diseños son los planos de un sitio o aplicación. Nos dicen cómo está conformada la estructura y qué es cada elemento. Permiten que todas las personas involucradas en el proyecto tengan una clara referencia de cómo se espera que quede armado el sitio o la aplicación.



2.VSC y carpetas

Es un IDE (entorno de desarrollo integrado) desarrollado por Microsoft. Cuando hablamos de un IDE, nos referimos a un conjunto de herramientas diseñadas para facilitarnos la creación y el desarrollo de nuestros programas o aplicaciones.

Entorno



Menú superior



Nos permite acceder a todas las funcionalidades de VS Code: crear nuevos archivos, guardarlos, edición de nuestro contenido, cambiar vistas, abrir terminales y mucho más.

Menú lateral (Barra de actividad/Activity bar)

Nos permite acceder rápidamente a las funcionalidades más utilizadas de VS Code. Repasémoslas en orden de arriba hacia abajo:



Explorador (Explorer): nos da acceso a visualizar nuestra estructura de carpetas y archivos, además de los archivos que estamos editando (*open editors*).

Buscar (Search): nos permite buscar texto dentro de nuestros archivos, también tiene la función de buscar y reemplazar.

Control de fuente - versiones (Source control): nos va a permitir comparar nuestra versión local con la versión en la nube.

Ejecutar (Run): nos permite correr y debuggear código.

Extensiones (Extensions): nos muestra las extensiones (funcionalidades agregadas) que tenemos instaladas y nos permite buscar e instalar nuevas.

Barra de estado (Status bar)

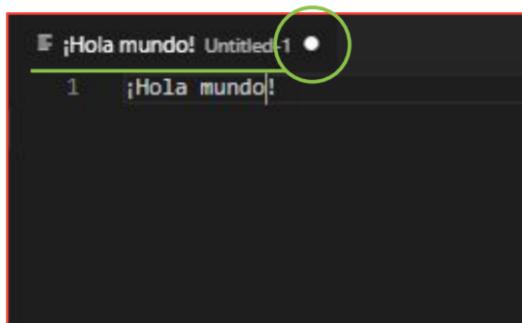
Nos da información del archivo que estamos editando.

Ln 1, Col 1	Spaces: 4	UTF-8	Plain Text
Nos indica en qué fila y columna de nuestro archivo estamos (a la fila la solemos llamar Línea y la columna es el número de carácter).	Espacios (Spaces): nos dice cuántos espacios estamos utilizando cuando tabulamos para indentar nuestro código.	Formato de codificación (Encoding)	Lenguaje del editor (Editor language): nos indica en qué lenguaje estamos trabajando.

Archivos guardados (o no)

Cada vez que modificemos un archivo, **VS Code** nos indicará que hay cambios sin guardar agregando un punto junto al nombre del archivo.

Es importante que todos los cambios estén guardados para poder verlos en acción.



Atajos/Shortcuts

Nos podemos ayudar muchísimo con algunos atajos que nos provee **VS Code** (¡estos son solo algunos de muchos!):

	(Windows / Linux)	(Mac)
Nuevo archivo.	Ctrl+N	Command+N
Abrir archivo.	Ctrl+O	Command+O
Guardar archivo.	Ctrl+S	Command+S
Guardar archivo como (mismo archivo con distinto nombre).	Ctrl+Shift+S	Command+Shift+S
Buscar en archivo (si estoy en el archivo).	Ctrl+F	Command+F
Buscar y reemplazar en archivo (si estoy en el archivo).	Ctrl+H	Command+H
Abrir terminal.	Ctrl+Ñ	Command+Ñ

Estructura de archivos

Trabajar con códigos, estilos e imágenes es parte natural de nuestro proceso de desarrollo, pero ¿cómo comenzamos a organizar nuestros archivos?

Si bien esta pregunta no tiene una respuesta 100% definitiva, sí podemos pensar en una manera estándar que permita organizarnos de la mejor manera posible. Es por este motivo que, a continuación, compartimos una estructura de archivos sugerida para optimizar mucho más el proceso de programar.

Estructura de archivos recomendada:

root - Carpeta raíz de nuestro proyecto. Suele llamarse **Public**

css - Carpeta con las hojas de estilos .css)

img - Carpeta de imágenes

(acá van todos los archivos de imagen (jpg, gif, png) que utilicemos en nuestro proyecto)

index.html - Página de inicio de nuestro proyecto. (no necesita estar dentro de otra carpeta, está en root)

Con dicha estructura, vamos a poder separar por responsabilidades cada uno de los elementos de nuestro proyecto y, de esta manera, gestionar mejor nuestro trabajo y el del equipo. Mantener una estructura similar en todos nuestros trabajos nos facilita las tareas de planificación y mantenimiento.

3. Navegadores y Dispositivos

Navegadores

Un navegador es un programa que tenemos, o podemos instalar, y abrir en nuestra computadora. A través del mismo podemos acceder y navegar por internet para visitar páginas, documentos, entre otros, gracias a la interpretación previa que hace de estos.

Compatibilidad

La forma en la que el navegador interpreta y muestra los archivos HTML con los estilos de los archivos CSS está establecida por el consorcio W3C (World Wide Web Consortium), que es la organización de estándares de Internet.

Conocer los distintos tipos de navegadores, así como sus incompatibilidades, es importante a la hora de desarrollar una página web. No todos los usuarios de un sitio web visitan la página con el mismo navegador que los creadores utilizan.

Tener esto en cuenta es muy importante para evitar que haya usuarios que abandonen nuestro sitio porque no se puede visualizar correctamente.

Una de las características que han hecho a Google Chrome tan popular es que es uno de los más completos navegadores en cuanto a respetar los estándares web.

Responsive

Cuando utilizamos el término “responsive” (adaptable), nos referimos principalmente a “responsive design” (diseño web adaptable). Esto significa hacer que un sitio web sea accesible y adaptable en todos los dispositivos: tablets, celulares, pantallas grandes.

CLASE III

Introducción a FIGMA.

Qué es

Figma es una herramienta de prototipado, con la cual podemos crear una interfaz de manera colaborativa. Está principalmente creada para realizar interfaces de apps mobile o sitios web, pero no quita que podamos realizar cualquier tipo de diseño. Podemos usar Figma sin la necesidad de bajar una aplicación, esto quiere decir que se puede utilizar mediante un navegador web. Sin embargo, si quisieramos utilizar Figma de manera offline, podemos descargar la aplicación, tengamos en cuenta que esta versión es compatible con Windows y Mac OS, pero no con Linux.

Para qué sirve

Figma es una aplicación basada en navegador o aplicación para diseñar UI/UX que cuenta con excelentes herramientas de diseño, creación de prototipos y generación de código. Actualmente es la herramienta líder en la industria para diseñar interfaces y cuenta con características sólidas que respaldan a los equipos que trabajan en cada fase del proceso de diseño.

Diseño y prototipado en Figma

Como desarrolladores, vamos a necesitar una guía, un boceto, un wireframe para poder codear nuestra web, y si nos ponemos a ver cómo se resolvía esto en otros tiempos, era papel y lápiz o bien una imagen, donde nosotros visualmente teníamos que resolver medidas, tratar de afinar los colores que usó el diseñador y demás formatos en la imagen.

Figma viene a resolver esto, un diseñador UX/UI nos va a facilitar un diseño del cual nos va a brindar qué tipografía usa, colores, tamaño de fuente y demás cosas que necesitemos.



No, un desarrollador por se no debe ser diseñador.



Es importante saber usarlo para conocer qué colores usamos, fuentes, tamaños y todo los elementos que necesitemos para el bocetado de nuestra app o sitio.

CLASE IV

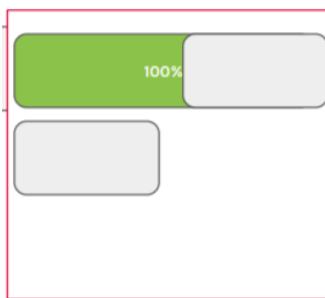
Introducción a HTML

En esencia, podríamos decir que HTML es el lenguaje que nos permitirá darle estructura a nuestros sitios web. Si miramos más profundo, HTML es mucho más que eso, ya que también nos permite darles un sentido a los componentes del sitio y permitir que sistemas como los buscadores entiendan su estructura y puedan mostrar su contenido de manera inteligente.

Cada **elemento** en el lenguaje HTML produce **un componente visual** en el navegador.

Flow de un sitio web

Si un elemento ocupa toda la pantalla, el siguiente colapsará y caerá al primer lugar que tenga disponible.



Algunos elementos que vamos a incorporar en nuestros sitios ya ocupan el 100% del ancho de nuestra pantalla, es por eso que veremos que casi todos los elementos se colocan uno debajo del otro.

Algunos elementos, por defecto, van a ocupar solo su contenido.



Etiquetas, atributos y estructura básica

HTML quiere decir **Hyper Text Markup Language**.

Sintaxis de un elemento

<h1>

...

</h1>

Etiqueta de apertura

Indica el comienzo de un elemento. Siempre debe iniciar con el símbolo de menor < y terminar con el signo de mayor >. Dentro debe ir el **nombre** de la etiqueta.

Etiqueta de clausura

Indica el final de un elemento. Siempre debe iniciar con el símbolo de menor seguido de la barra diagonal </ y terminar con el signo de mayor >. El **nombre** va adentro.

Atributos

Son configuraciones adicionales de los elementos que ajustan su comportamiento de diversas formas.

Contenido

Todo aquello que escribamos entre las etiquetas de apertura y cierre de un elemento, conformarán su contenido.

html <h1 align="center"> ... </h1>

Valores

Nos permiten definir las configuraciones. Siempre van a estar escritos entre comillas " " y luego de un signo de igual.

Estructura básica de un documento html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
  </head>
  <body>
    <p>Lorem ipsum dolor sit amet, consectetur
       adipisicing elit. Tenetur deserunt molestiae
       numquam veritatis ea ut praesentium explicabo
       atque maxime a eaque, aut id consequuntur. Et
       nemo non perspiciatis eum!</p>
  </body>
</html>
```



Introducción a la semántica

Cuando realizamos nuestro código es muy importante que desde las etiquetas empecemos a poner un orden semántico. El objetivo es que, a través de las mismas, y a simple vista, podamos identificar un encabezado, la sección principal del sitio, distintas secciones, artículos, cuál es el pie de página o una barra de navegación.

Utilizar etiquetas semánticas nos permite generar un código más entendible, tanto para los humanos como para los buscadores.

Las etiquetas semánticas de bloque

Las etiquetas que veremos a continuación se comportan todas exactamente igual que un **<div>**. En otras palabras, son todas etiquetas de bloque. La diferencia está en que, según el nombre y el lugar donde ubicemos la etiqueta, esta nos dará contexto sobre la información que hay dentro. Esto permitirá que tanto personas que lean el código, como buscadores, entiendan la intención de la estructura y el contenido que generamos.

El encabezado

La etiqueta **<header>** nos sirve para generar la cabecera del documento, o incluso también para generar la cabecera de una sección de contenido. Tradicionalmente, aquí van cosas como el logotipo de la web, la barra de navegación, los enlaces a las redes sociales e incluso un pequeño campo de búsquedas rápidas.

```
<header>
    <!-- Logotipo -->
    <!-- Barra de navegación -->
    <!-- Barra de búsqueda -->
    <!-- Enlaces a redes sociales -->
</header>
```

La navegación

La etiqueta **<nav>** nos sirve para generar una barra de navegación, sea la navegación principal o una navegación alternativa. Dentro de esta etiqueta, el árbol de navegación se suele implementar a través de listas desordenadas, elementos de listas y enlaces o hipervínculos.

```
<nav>
    <ul>
        <li><a href="#">Inicio</a></li>
        <li><a href="#">Quiénes somos</a></li>
        <li><a href="#">Servicios</a></li>
        <li><a href="#">Contacto</a></li>
    </ul>
</nav>
```

El pie de página o de sección

La etiqueta **<footer>** nos sirve para generar el pie de página principal del documento o el pie de página de una sección de contenido. Tradicionalmente, aquí van cosas como los derechos reservados y algunos enlaces adicionales de la web.

```
<footer>
    <p>Todos los derechos reservados</p>
    <nav>
        <ul><a href="#">Términos y condiciones</a></ul>
        <ul><a href="#">Mapa de sitio</a></ul>
        <ul><a href="#">Información legal</a></ul>
    </nav>
</footer>
```

Las secciones

La etiqueta **<section>** nos permite definir una sección de contenido. Si quisieramos crear un breve apartado sobre un producto o servicio, esta etiqueta sería una excelente

opción. Aquí también podemos usar las etiquetas de encabezado y pie.

```
<section>
  <header>
    <h2>Producto destacado</h2>
  </header>
  <!-- Contenido del producto -->
  <footer>
    <a href="#">Continuar leyendo...</a>
  </footer>
</section>
```

Los artículos

La etiqueta **<article>** nos permite definir una pieza de contenido independiente. Es decir, contenido que podría funcionar por sí solo sin necesidad de todo lo que lo rodea: un producto, un servicio, una noticia, etcétera. Generalmente los vamos a ver dentro de secciones, pero no es obligatorio.

```
<section> <!-- Productos destacados -->
  <article>
    <!-- Contenido del producto -->
  </article>
  <article>
    <!-- Contenido del producto -->
  </article>
</section>
```

Elementos de línea y de bloque

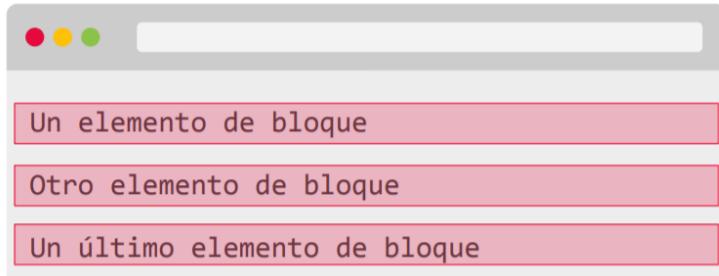
Si bien hay muchos elementos en las páginas web, estos se pueden agrupar en los siguientes grupos:

1. Elementos de bloque
2. Elementos de semi-bloque
3. Elementos ocultos
4. Elementos en línea

Cuando hablamos de la visualización y el comportamiento de los elementos de HTML, hay dos grandes grupos: los elementos **de bloque** y los **de línea**.

Elementos de bloque

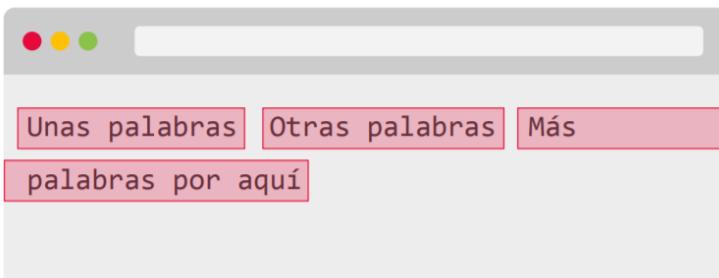
Los elementos de bloque intentan ocupar el 100% del ancho del sitio. Visualmente generan un salto de línea. Esto se da porque, al ocupar todo el ancho disponible, no dejan espacio para que entre otro elemento.



Las etiquetas **<div>** son un ejemplo de **etiquetas de bloque** muy utilizadas, ya que permiten generar divisiones en nuestro sitio.

Elementos de línea

Los elementos de línea determinan su tamaño en base al contenido que tengan. Eso quiere decir que pueden convivir uno al lado del otro en el ancho total del sitio. Si el contenido excede la línea, continúa en la de abajo.



Las etiquetas **** son de línea y se suelen usar para contener porciones de texto y así poder determinarle un estilo independiente al bloque de código al que pertenecen.

La propiedad display

Mediante la propiedad display de CSS podemos cambiar la disposición del elemento que queramos. Los valores que recibe son block, inline, inline-block y none.

```
.en-línea { display: inline; }
.de-bloque { display: block; }
.de-bloque-en-línea { display: inline-block; }
.oculto { display: none; }
```

Disposiciones de elementos

Inline

Define un elemento con comportamiento en línea. No recibe algunas propiedades del modelo de caja.

Block

Define un elemento con comportamiento de bloque. Puede recibir propiedades del modelo de caja.

Inline-block

Define un elemento con comportamiento de semi-bloque. Puede recibir propiedades del modelo de caja, y también comparte propiedades de elementos de línea.

None

Oculta un elemento. No lo elimina de la estructura de HTML, solo desaparece de la vista.

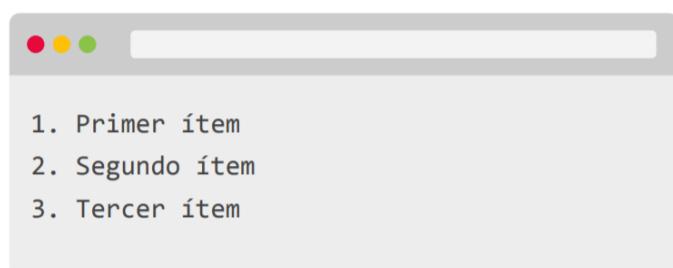
CLASE V

Listas en HTML

1. Listas ordenadas

Las listas ordenadas nos permiten enumerar ítems de manera consecutiva. Por defecto van a empezar en el número 1 y se irán incrementando con cada ítem nuevo.

```
<ol>
    <li>Primer ítem</li>
    <li>Segundo ítem</li>
    <li>Tercer ítem</li>
</ol>
```



Atributo → type

Nos permite cambiar el tipo de viñeta de la lista.

```
html      <ol type="1"> ... </ol>
```

Valores

- Numérico (1)
- Alfabético (A)
- Numérica romana (I)

Atributo → start

Nos permite definir dónde va a empezar nuestra numeración.

```
html <ol start="20"> ... </ol>
```

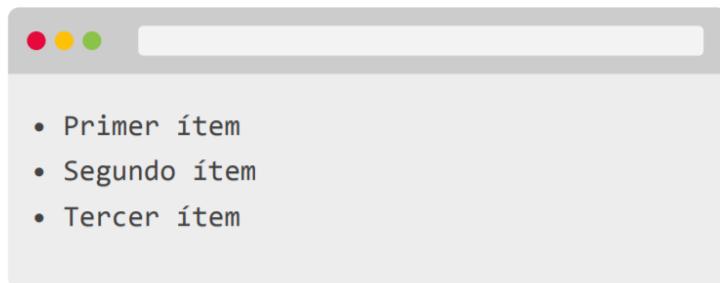
Valor

Puede ser cualquier número positivo o negativo.

2. Listas desordenadas

Las listas desordenadas también nos permiten listar ítems. Por defecto va a generar una viñeta tipo “bolita” por cada ítem nuevo que se agregue.

```
<ul>  
  <li>Primer ítem</li>  
  <li>Segundo ítem</li>  
  <li>Tercer ítem</li>  
</ul>
```



Atributo → type

Nos permite cambiar el tipo de viñeta de la lista.

```
html <ul type="disc"> ... </ul>
```

Valor

- Disc (●)
- Circle (○)
- Square (■)
- None

3. Listas anidadas

Las listas anidadas nos permiten crear varios niveles de jerarquía y organización. Las podemos anidar como deseemos y generar los niveles que necesitemos.

```

<ul>
  <li>Recordar para el viaje:
    <ol>
      <li>DNI</li>
      <li>Pasajes</li>
    </ol>
  </li>
  <li>Llamar a Assist Card</li>
  <li>Adaptador para celular</li>
</ul>

```

Dentro de un o un solo puede haber elementos .



Las listas se pueden anidar agregando un o dentro de un .

- Recordar para el viaje
 - 1. DNI
 - 2. Pasajes
- Llamar a Assist Card
- Adaptador para celular

Configuración de caracteres

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
  </head>
  <body>
    <p>Lorem ipsum dolor sit amet, consectetur
       adipisicing elit. Tenetur deserunt molestiae
       numquam veritatis ea ut praesentium explicabo
       atque maxime a eaque, aut id consequuntur. Et
       nemo non perspiciatis eum!</p>
  </body>
</html>

```

Configuración de la página

```

<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Hello world</title>
  </head>
  <body>
    <p>Lorem ipsum dolor sit amet, consectetur
       adipisicing elit. Tenetur deserunt molestiae
       numquam veritatis ea ut praesentium explicabo
       atque maxime a eaque, aut id consequuntur. Et
       nemo non perspiciatis eum!</p>
  </body>
</html>

```

Configuración de caracteres

Anatomía del código

Atributo → charset

Permite definir la codificación de caracteres a utilizar. HTML funciona por defecto para el inglés, cuando queremos utilizar caracteres de otros idiomas, debemos especificarlos.



Etiqueta → meta

Permite definir propiedades de la página que no pueden definirse en otras etiquetas, como `<title>`, `<link>` o `<style>`.

Suelen ser datos sobre el contenido, su descripción, autor, caracteres, etcétera.

Valor → utf-8

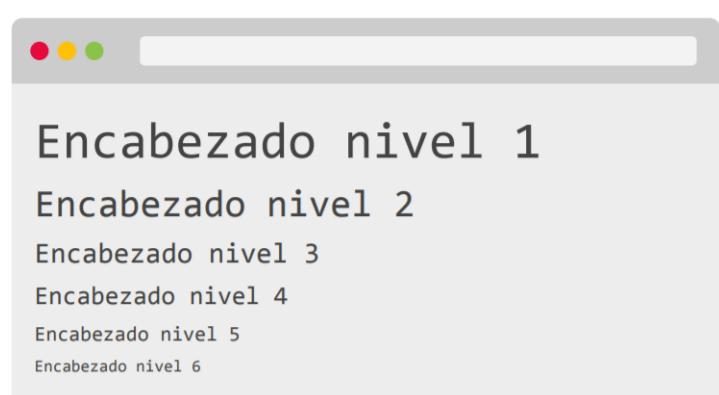
El estándar Unicode (UTF-8) está diseñado para permitir la correcta visualización de los caracteres utilizados por la mayoría de los lenguajes y disciplinas técnicas del mundo moderno.

Etiquetas de Texto

Elementos de encabezado

Los elementos de encabezado implementan seis niveles de encabezado del documento, `<h1>` es el más importante, y `<h6>` es el menos importante. Un elemento de encabezado describe brevemente el tema de la sección que presenta. Son elementos de bloque.

```
html <h1>Encabezado nivel 1</h1>
      <h2>Encabezado nivel 2</h2>
      <h3>Encabezado nivel 3</h3>
      <h4>Encabezado nivel 4</h4>
      <h5>Encabezado nivel 5</h5>
      <h6>Encabezado nivel 6</h6>
```



El elemento `<h1>`, por recomendación de la W3C, solo debe ser utilizado **una vez** por documento HTML.

Elementos de párrafo

Los elementos de párrafo `<p>` nos permiten distribuir el texto en párrafos. Podemos usar tantos como necesitemos. Son elementos de bloque.

```
html      <p>Este es un párrafo, puede tener todo el texto que  
          necesitemos.</p>  
          <p>El navegador se encargará de agregar espacio vertical  
          entre cada uno de los párrafos que escribamos.</p>
```

Rutas, hipervínculos e imágenes

1. Qué es una ruta

Es una dirección o camino (también conocido con el término inglés path), que le va a permitir al navegador encontrar un recurso. Ese recurso puede ser otra página web, una imagen, un video o cualquier otro tipo de archivo. En el caso de los enlaces, la ruta indica la dirección a la que tiene que llevarnos el navegador cuando pulsamos sobre él.

Ruta absoluta

<https://www.google.com>

Ruta relativa

[..../imagenes/perfil.jpg](#)

Ejemplo de ruta absoluta: Si tenemos una imagen que está alojada en una URL, podemos acceder a ella sin importar el lugar en el que esté navegando en ese momento. Su ruta, es decir, el acceso hacia ese recurso, es siempre el mismo.



<https://unsitioweb.com/imagenes/imagen1.jpg>

Ejemplo de ruta relativa: Para acceder a una imagen que se encuentra en la misma carpeta que el archivo original, la ruta simplemente debe hacer referencia al nombre del otro archivo ya que ambos archivos comparten el mismo lugar.



ruta desde **unArchivo.js** hacia **unalImagen.jpg**:

unalImagen.jpg

Para acceder a una imagen que se encuentra en una **carpeta inferior** a la del archivo original deberíamos descender un nivel. Eso lo hacemos con la barra /.



Si hubiese **más carpetas inferiores**, solo se debería ir accediendo una a una utilizando las **barras /**, hasta llegar a la imagen.



Para acceder a una imagen que se encuentra en una carpeta **a la misma altura** a la del archivo original, deberemos salir de la actual y luego entrar en la correspondiente. Para ir un nivel hacia atrás, usamos dos puntos seguidos ..



2. Hipervínculos o enlaces

A través de la etiqueta `<a>` vamos a poder crear nuestros enlaces.

Ruta

Aquí podremos especificar una ruta absoluta (como en el ejemplo) o una relativa.

```
html <a href="https://www.google.com">Vamos a google</a>
```



atributo → href

Se utiliza para indicar el **destino** al que apunta nuestro enlace.



Texto

Aquí podremos escribir el contenido que va a visualizar el usuario.

Tipos de enlaces

Externos

Sus rutas están fuera de nuestro sitio, por lo tanto, en general serán absolutas.

```
html <a href="https://www.google.com">Vamos a google</a>
```

Locales

Sus rutas están dentro de nuestro sitio. Se recomienda que sean relativas siempre que sea posible.

```
html <a href="inicio.html">Inicio</a>
```

Anclas

Sirven para hacer referencia a un fragmento o elemento dentro de una página; como puede ser una sección o un titular y pueden combinarse con las anteriores. Inician con el carácter `#` y hacen referencia a la propiedad `id`.

```
html <a href="#biografia">Biografía</a>
```

```
html <a href="https://www.sitio.com/#contacto">Contacto</a>
```

```
html <a href="nosotros.html#nuestro-equipo">Nuestro equipo</a>
```

Correo

Al hacer clic en ellos, se abrirá el programa de correo que tengamos como predeterminado para enviar un correo a esa dirección de e-mail.

```
html <a href="mailto:user@server.com">Enviar mensaje</a>
```

Teléfono

Parecido al caso anterior, si estamos usando nuestro smartphone, se iniciará una llamada a ese número o aparecerá listo para llamar.

```
html <a href="tel:1145678899">Inicio</a>
```

3. Imágenes

Fuente de una imagen

Dentro de nuestro documento HTML podemos agregar imágenes a través de la etiqueta ****. Esta etiqueta nos permite invocar las imágenes, es decir, hacer referencia al lugar donde están alojadas para que aparezcan en el navegador.

```
html 
```

atributo → src

Ruta

Se utiliza para indicar el destino en donde está alojada nuestra imagen.

Aquí podremos especificar una ruta absoluta (como en el ejemplo) o una relativa.

Texto alternativo

Nos ayuda a darle una descripción a la imagen y también nos permite:

- Ayudar a que los buscadores entiendan la imagen.
- Mostrar un texto si la imagen no carga
- Darle contexto a las personas con discapacidades visuales.

```
html 
```

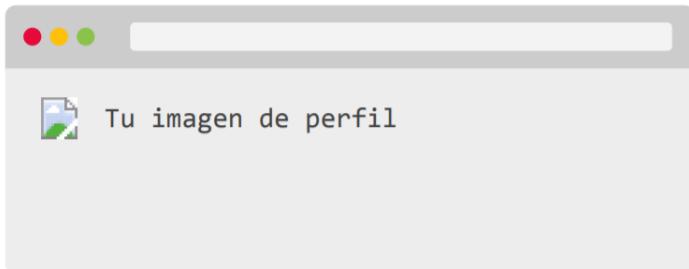
atributo → alt

Texto

Este atributo nos permite especificar el texto alternativo.

Debe contener una descripción corta de la imagen que deberíamos estar viendo. Soporta 125 caracteres.

Como dijimos, en caso de que por alguna razón la imagen no cargue, el navegador podrá mostrar el texto alternativo. Según qué navegador estemos usando, puede verse distinto.



Ancho de la imagen

Nos permite indicar el ancho de nuestra imagen. No es obligatorio. Los valores pueden ser tanto en **píxeles** (solo escribimos el número) como en **porcentajes** (con el símbolo % al final).

```
html      
```



atributo → width

Este atributo nos permite especificar el ancho de la imagen.

Alto de la imagen

Nos permite indicar el alto de nuestra imagen. No es obligatorio. Los valores pueden ser tanto en **píxeles** (solo escribimos el número) como en **porcentajes** (con el símbolo % al final).

```
html      
```



atributo → height

Este atributo nos permite especificar el alto de la imagen.

Accesibilidad

La accesibilidad es el conjunto de medidas, que deben tomarse durante el diseño y desarrollo del sitio, para que el producto final sea comprendido, utilizado y alimentado por todos los usuarios sin importar si tienen alguna discapacidad, es decir, que todos nuestros visitantes y/o usuarios puedan disfrutar de nuestro producto sin distinción alguna.

Algunos mitos que suelen plantearse:

- Es un servicio que se paga aparte.
- Apunta a un bajo porcentaje de receptores.
- No representa un aporte social o económico.
- Es solamente parte de las buenas prácticas.

Los límites de accesibilidad que pueden presentarse se agrupan en:

- Visuales.
- Motrices.
- Auditivos.
- Cognitivos.
- Inmigrantes digitales (personas no son nativos digitales, a quienes les puede resultar difícil utilizar tecnología).

¿Cómo hacer sitios accesibles?

Podemos plantear algunas estrategias dependiendo del tipo de limitación que queramos resolver. Veamos el siguiente cuadro.

Limitante	Propuesta superadora
Dificultades visuales	Transcripción de audio
	Tamaño del texto amplio
	Utilización de contrastes adecuados
Dificultades auditivas	Videos con subtítulos
Problemas motrices	Tamaño de botones y áreas activas adecuados para poder utilizar el mouse
	Interactuar con el sitio tanto a través del teclado como del mouse
Dislexia	Contenidos acompañados de ilustraciones y diagramas
Problemas cognitivos	Utilización de iconos representativos

Estos son solo algunos ejemplos de medidas que podemos incorporar. Para conocer más, podemos visitar el siguiente link:

<https://www.w3c.es/Traducciones/es/WAI/intro/accessibility>

Buenas prácticas

Aunque aún no hemos profundizado en estos temas, queremos plantear algunas propuestas de buen uso que serán muy útiles más adelante y podremos volver a consultar.

Buen uso de HTML

Utilización de:

- Encabezados
- Textos alternativos de imágenes
- Label en los formularios
- Enlaces descriptivos
- No resaltar contenido sólo por colores

Buen uso de CSS

Utilización de:

- Texto y fondo con imagen de contraste
- Utilizar medidas relativas de fuente
- Familia de texto alternativas
- Utilizar media queries para los dispositivos de salidas

Existen extensiones para instalar en nuestros navegadores que nos permiten ver nuestro sitio como lo vería una persona con alguna dificultad. Este es el enlace que nos permite instalar una de ellas, llamada **funkify**, en Chrome:

<https://chrome.google.com/webstore/detail/funkify>

GitHub pages

GitHub cuenta con una característica que nos permite crear o publicar una página web para nuestro proyecto, llamada GitHub Pages.

Es un servicio que nos ofrece GitHub para publicar de una manera muy sencilla páginas web. Disponemos de la opción de generar una URL de nuestro repositorio accediendo de manera sencilla desde la configuración.

1 Vamos a nuestros repositorios.

2 Seleccionamos el repositorio en que vamos a generar la URL de Github Pages, debe tener index.html en la raíz.

3 En la barra de acciones hacemos clic en Settings.

4 Buscamos la opción GitHub Pages. Branch lo dejamos en main, root en / y hacemos clic en Save.

5 ¡Voilà! Ya tenemos nuestro link generado.



Git paso a paso

Comandos paso a paso para crear un repositorio local

```
>_ git init // crea el repositorio
```

```
>_ git config user.name "nombreUsuario" // agrega nuestra identidad
```

```
>_ git config user.email "emailUsuario" // agrega nuestro e-mail
```

```
>_ git remote add origin http://... // apunta al repositorio remoto
```

Comandos paso a paso para subir cambios

```
>_ git add . // agrega todos los archivos
```

```
>_ git commit -m "mensaje" // commitea los cambios hechos
```

```
>_ git push origin master // envía los cambios al repositorio remoto
```

```
>_ git status // realiza un seguimiento de los estados de los archivos
```

otros comandos

```
>_ git status // realiza un seguimiento de los estados de los archivos
```

```
>_ git pull // descarga los cambios que existen en el repositorio remoto
```

CLASE VII

Introducción al lenguaje de estilos CSS

CSS quiere decir... Cascading Style Sheet

¿Para qué sirven las hojas de estilo? Estas sirven para estilizar nuestro contenido HTML. Con CSS podemos cambiar colores, fondos, tipografías, anchos, altos, y mucho más. Así como también generar animaciones y transiciones. Contamos con 3 métodos para vincular nuestros archivos CSS con el documento HTML.

Vinculación externa (recomendada)

Esta es la más utilizada ya que podemos escribir todos nuestros estilos en un archivo CSS y vincularlos al HTML usando la etiqueta `<link>` dentro del `<head>` de nuestro documento.

Atributo + valor

Indica qué relación hay entre los documentos a enlazar. El valor siempre será el mismo.

```
html      <link href="css/estilos.css" rel="stylesheet">
```

Atributo + valor

Ruta de la ubicación de mi hoja de estilo.

Vinculación interna (no recomendada)

A través de la etiqueta `<style>` dentro del `<head>`.

```
html      <head>
          <meta charset="UTF-8">
          <style>
            body {background: blue}
          </style>
        </head>
```

Vinculación en línea (no recomendada)

Usando el atributo `style` en cada elemento de nuestro HTML.

```
html      <p style="color: red">...</p>
```

Las reglas de CSS

Una regla de CSS es un conjunto de instrucciones que se aplican a un elemento determinado para agregarle estilos.

Selector

Indica sobre qué elemento o elementos aplicaremos la regla.



propiedad

valor

Define qué propiedad del elemento o de los elementos vamos a modificar.

Especifica el comportamiento de la propiedad.

¿Cómo se escriben? Primero escribimos el o los selectores, seguidos de las llaves {} que indican el comienzo de las reglas. Cada regla se compone de una propiedad y su correspondiente valor, separados por **dos puntos :**. Finalmente, podemos tener más de una regla por elemento y estas **se separan con punto y coma ;**.

```
css      body {  
           background-color: purple;  
           font-family: sans-serif;  
           text-align: center;  
         }
```

Las propiedades de CSS

Existen muchas propiedades de CSS que nos permiten manipular los elementos del HTML a nuestro antojo. Si tuviéramos que agrupar las propiedades, los grupos serían más o menos así:

- Tipografías
- Fondos
- Tamaños
- Posicionamiento
- Visualización
- Comportamiento
- Interfaz
- Otros

Los selectores de CSS

Selectores de ID

Este selector va a atrapar al elemento HTML que tenga asignado el atributo **id** con el valor correspondiente. Se recomienda usar nombres únicos para cada elemento y no repetirlos a lo largo del documento.

```
html <h3 id="saludo">¡Hola!</h3>
```

Para llamarlo desde el CSS usamos el **#** seguido del **nombre del ID**.

```
css #saludo { color: blue }
```

Selectores de clase

Este selector va a atrapar al elemento HTML que tenga asignado el atributo **class** con el valor correspondiente. Podemos asignarle la cantidad de clases que queramos a un mismo elemento. Para hacerlo, solo hace falta separarlas con un espacio.

```
html <h3 class="noticia destacada">Una noticia</h3>
```

Para llamarlo desde el CSS usamos el **.** seguido del **nombre de la clase**.

```
css .noticia { font-size: 22px }
```

Selectores de etiqueta

Este selector va a atrapar al elemento HTML con el **mismo nombre de etiqueta** que llamemos desde nuestro CSS.

```
html <p>Ad lorem ipsum dolor sit amet</p>
```

Para llamarlo desde el CSS usamos el **nombre de la etiqueta**.

```
css p { color: gray }
```

Selectores combinados

Como su nombre lo indica, son la combinación de todos los anteriores, también podemos usar varios selectores del mismo tipo. En este ejemplo combinamos un selector de etiqueta con uno de clase.

```
html <h2 class="subtitulo">Un subtítulo</h2>
```

Para llamarlos desde el CSS solo hace falta agregar un selector al lado del otro, cada uno con la sintaxis que le corresponda. Recordemos que van sin espacios.

```
css h2.subtitulo { color: yellow }
```

Selectores descendentes

Estos selectores sirven para agregar especificidad. En el ejemplo vamos a atrapar al elemento li que esté dentro del ul con el id lista. Para llamarlos desde el CSS escribimos los selectores separados por un espacio (el de la derecha siempre será el que está dentro del de la izquierda).

```
html <ul id="lista">
      <li>Primer ítem</li>
    </ul>
```

```
css ul#lista li { text-align: center }
```

El CSS siempre va a priorizar a los selectores más específicos para aplicar los estilos.

FUENTES

font-family

Permite elegir la **familia tipográfica** que queremos usar. Como valor recibe el nombre de la tipografía que queramos usar. Para que funcione, la tipografía debe existir en la computadora del usuario o usar una webfont. Podemos poner más de una tipografía, separando las adicionales por comas. En caso de que la primera no esté disponible, se cargará la segunda y así sucesivamente.

```
css p {
  font-family: Arial, sans-serif;
}
```

font-size

Permite definir el **tamaño tipográfico**. Recibe un valor numérico acompañado de la unidad de medida. Las unidades de medida más habituales suelen ser: px, em y rem.

```
css      p {  
          font-size: 23px;  
        }
```

font-style

Define el **estilo de la tipografía**. Recibe los valores italic, normal y oblique. Para algunos elementos, como , el valor por defecto será italic.

```
css      p {  
          font-style: normal;  
        }
```

font-weight

Define el **peso** de la tipografía. Recibe los valores bold, lighter, normal, entre otros. También puede recibir un valor numérico que se irá incrementando de 100 en 100. Para algunos tags el valor por defecto será bold.

```
css      p {  
          font-weight: 500;  
        }
```

text-align

Permite definir la **alineación** del texto. Los valores que recibe son center, left, right, inherit y justify. El valor por defecto para todos los elementos es left.

```
css      p {  
          text-align: justify;  
        }
```

text-decoration

Permite elegir un tipo de **decoración** para el texto. Recibe los valores line-through, underline, overline y none. Para algunos elementos, como los enlaces, el valor por defecto será underline.

```
css      p {  
          text-decoration: underline;  
        }
```

line-height

Permite definir el alto de cada línea de textos. Esto también suele llamarse **interlineado**. Recibe un valor numérico acompañado de la unidad de medida y, por lo general, está relacionado con el tamaño de la letra o font-size.(al tamaño del font-size se le adiciona como mínimo 8px para obtener un interlineado estandar)

```
css   p {  
      line-height: 20px;  
}
```

COLORES

Formatos de color

Los siguientes formatos se pueden aplicar en cualquier propiedad de CSS que reciba color:

- Nombre: purple VER: <https://htmlcolorcodes.com/color-names/>
- Hexadecimal #f05331
- RGB rgb(255, 100, 50)
- RGBA rgba(122, 50, 125, 0.5). El último número representa la opacidad que tendrá el elemento. Va del 0 al 1 y mientras menor el número, mayor la transparencia.

Color de texto

El atributo **color** nos permite asignarle un color al texto de un elemento. Recibe como valor cualquiera de los formatos de color permitidos.

```
css   h2 { color: purple }
```

```
css   h2 { color: #3459ff }
```

```
css   p { color: rgb(12, 34, 32) }
```

Color de fondo

El atributo `background-color` nos permite asignarle un color de fondo a un elemento. Recibe como valor cualquiera de los formatos de color permitidos.

```
css p { background-color: tomato }
```

```
css p { background-color: #3459ff }
```

```
css p { background-color: rgb(12, 34, 32) }
```

Opacidad

Mediante el atributo `opacity` le otorgamos transparencia a todo el elemento.

```
css p { opacity: 0.5 }
```



Valor

El valor representa el porcentaje de opacidad que le queremos dar al elemento.

Los valores van desde 0 a 1. Eso quiere decir que 0.5 representa una opacidad del 50%.

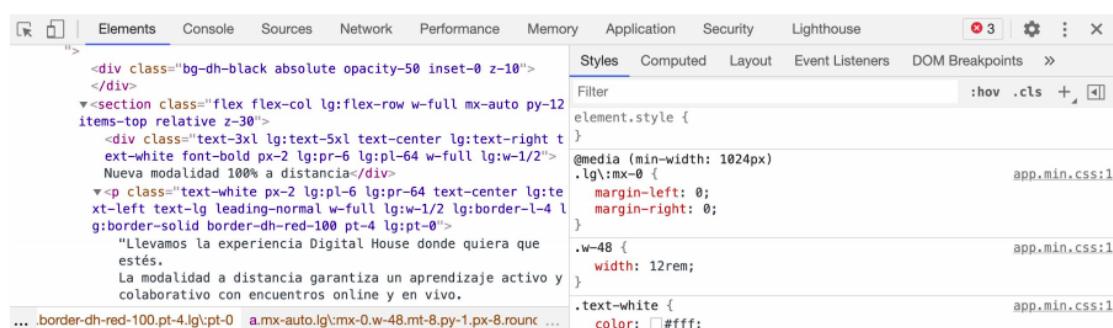
INSPECTOR DE PROPIEDADES (F12 para abrir en Chrome)

Utilizamos el Inspector para examinar y modificar el HTML y CSS de una página.

Para abrir el Inspector de elementos podemos hacerlo pulsando la tecla F12 o bien pulsando con el botón derecho en cualquier parte de la página y acceder a través de la opción “Inspeccionar”.

Pestaña Elements

Dentro de la pestaña “Elements” tendremos la consola del navegador para el código HTML y las reglas CSS



Explorar HTML: Al pasar el cursor por encima del código HTML nos ilumina el elemento seleccionado.

Editar HTML: Podemos editar las etiquetas HTML en el inspector haciendo clic con el botón derecho y seleccionando "Editar como HTML". Recordemos que esos cambios se realizan únicamente en nuestro navegador y al refreshar la página interpretará el código real de nuestro documento.

Explorar y editar CSS

La vista 'Estilos' muestra todos los estilos creados, incluidos aquellos que no se admiten o que no son válidos. Esto puede ayudarnos a entender por qué ciertos estilos no se están aplicando. También encontraremos en qué línea y documento CSS se encuentra esa regla. Se pueden editar en el navegador para probar cambios y mejoras.

The screenshot shows the 'Styles' tab in the Chrome DevTools. It lists several CSS rules from 'app.min.css:1':

```
element.style {  
}  
.w-48 {  
    width: 12rem;  
}  
.text-white {  
    color: #fff;  
}  
.text-center {  
    text-align: center;  
}  
.px-8 {  
    padding-left: 2rem;  
    padding-right: 2rem;  
}  
.py-1 {  
    padding-top: .25rem;  
    padding-bottom: .25rem;  
}
```

BUENAS PRÁCTICAS

Existen una serie de buenas prácticas que pueden orientarnos a escribir un código de calidad que sea limpio y ordenado. Veamos algunas sugerencias:

Crear primero tu HTML: Crear primero la estructura HTML te permite visualizar toda la página como un todo, y te permite pensar en tu CSS de una manera más organizada.

Comentar: No te olvides de comentar cada sección de tu documento.

```
***** clases generales*****/  
Estilos aquí_  
***** header *****/  
Estilos aquí_  
***** nav menu *****/  
Estilos aquí_
```

Nombrar correctamente los selectores: Para conseguir más claridad en el código y soporte en todos los navegadores conviene no comenzar el nombre de los selectores con mayúsculas, números ni caracteres especiales.

Separar las palabras mediante guiones o mediante mayúsculas: Elige una única manera de escribir el nombre de los selectores. Además, no utilices guiones bajos “_” u otros caracteres especiales ya que algunos navegadores no los soportan. Es recomendable seguir alguna de las siguientes opciones:

```
/* Opción 1: Palabras separadas por guiones */
.nombre-clase{
    color: orange;
}
/* Opción 1: Palabras separadas por mayúsculas */
.nombreClase{
    color: orange;
}
```

Utilizar nombres descriptivos en los selectores: Utiliza nombres que te permitan averiguar fácilmente a qué elemento le estás dando estilo.

```
/* Estilo del botón de la navbar*/
.nav-button {
    background: blue;
}
```

No utilices como nombre de un selector una característica visual: Al utilizar en el nombre de un selector una característica visual como el color, el tamaño o la posición y posteriormente, modificamos esa característica, también deberíamos cambiar el nombre del selector. Esto complica mucho el código ya que tendríamos que actualizar todas las referencias a ese selector en HTML

```
/* Selector con nombre que define la característica visual
del color */
.menu-red {
    background: red;
}
/* Utiliza mejor: */
.nav-menu {
    background: red;
}
```

Combinar elementos: Los elementos de una hoja de estilo a veces comparten propiedades. En lugar de volver a escribir el código anterior, ¿por qué no sólo combinarlos? Por ejemplo, los elementos h1, h2 y h3 podrían compartir la misma fuente y el mismo grosor:

```
h1, h2, h3 {  
    font-family: Arial;  
    font-weight: 700;  
}
```

Utilizar varias clases: A veces es beneficioso agregar varias clases a un elemento.

Orden de declaración: Para el orden de declaración resulta mucho más cómodo usar el orden alfabético para poder buscar luego una propiedad, por ejemplo si querés buscar un z-index sabes que siempre está al final de las declaraciones.

```
#menu-nav {  
    color: #fff;  
    float: left;  
    font-weight:bold;  
    height: 200px;  
    margin: 0;  
    padding: 0;  
    width: 150px;  
}
```

CLASE VIII

Colores de fondo

background-color

Nos permite asignarle un color de fondo a un elemento. Recibe como valor cualquiera de los formatos de color permitidos.

```
css      p { background-color: tomato }
```

```
css      p { background-color: #3459ff }
```

```
css      p { background-color: rgb(12, 34, 32) }
```

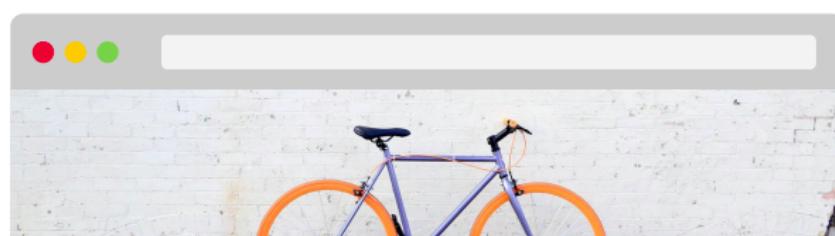
Imágenes de fondo

background-image

Nos permite asignarle una imagen de fondo al elemento, definiendo la ruta a través de la URL.

css

```
body { background-image: url('../img/bici.jpg') }
```

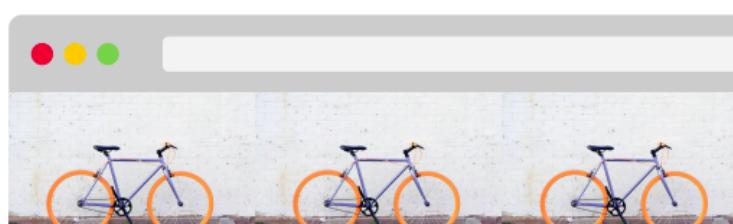


background-repeat

Nos permite controlar si se va a repetir, y de qué manera, la imagen dispuesta. Recibe los valores: repeat, no repeat, repeat-x, repeat-y, round y space.

css

```
body { background-repeat: repeat-x }
```



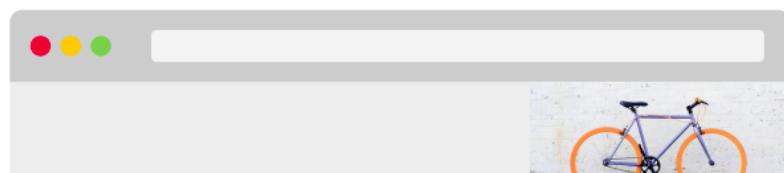
background-position

Nos permite mover la imagen dentro del elemento y decidir dónde colocarla. Recibe como valores tamaños en pixeles y porcentajes, así como también right, bottom, left, etcétera.

Podemos asignarle uno o dos valores. El primero para especificar la posición en el eje x y el segundo, la posición en el eje y.

css

```
body { background-position: right top }
```

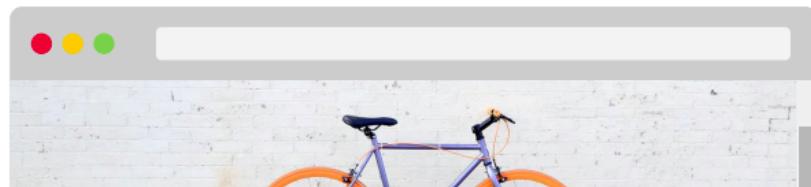


background-attachment

Nos permite establecer si la imagen de fondo se va a mover junto con la página al hacer scroll o si se va a quedar fija. Recibe como valor: fixed, scroll, inherit e initial.

css

```
body { background-attachment: fixed }
```



background-size

Nos permite establecer el tamaño de la imagen de fondo. Recibe como valor: contain, cover, inherit; así como también tamaños en pixeles y porcentajes, indicando con el primer valor el ancho y con el segundo, el alto.

css

```
body { background-size: 130px }
```



Fuentes genéricas, web y locales

Fuentes seguras (lista completa en https://www.w3schools.com/csSref/css_websafe_fonts.asp)

Es el conjunto de familias tipográficas que pueden ser usadas en cualquier página web sin problemas porque están preinstaladas en todos los sistemas operativos (linux, windows, IOS, Android) que utilizan nuestros dispositivos.

Brush Script MT Courier New Arial
Verdana Georgia Tahoma Trebuchet MS
Times New Roman Garamond Helvetica

Pero... ¿qué problemas podemos tener con las fuentes?



Que si el visitante no tiene instalada la fuente que elegimos para nuestro sitio puede que se vea todo muy feo.

Por lo tanto..



Debemos asegurarnos que al construir nuestro sitio la fuente esté disponible para todos los visitantes.

CSS definiendo familias tipográficas



Una buena práctica al seleccionar la fuente que utilizaremos, es **darle opciones al navegador** para que si no encuentra la ideal, pueda reemplazarla con otra que se ajuste a nuestro diseño.

Podríamos escribir una línea como la siguiente:

```
css p { font-family: Arial, Verdana, Garamond; }
```

Lo que le decimos al navegador es:

"Si el usuario tiene instalada la fuente **Arial**, le mostrás el párrafo en Arial, sino en **Verdana** y sino en **Garamond**".

Entonces... ¿cómo podemos asegurarnos de que nuestro sitio se vea bien sin complicarnos?



Las **fuentes genéricas** son un gran aporte en este sentido.

Podríamos cambiar la sentencia anterior por esta:

```
css p { font-family: Arial, Verdana, sans-serif; }
```

De esta manera nos cubrimos las espaldas, porque aunque el visitante no tenga la tipografía ideal, de todos modos verá el sitio muy parecido a nosotros.

En este caso, lo le decimos al navegador es:

"Mostrá el párrafo con **Arial** si está instalada, sino con **Verdana** y sino con cualquier otra tipografía **sans-serif** que tenga instalada".

Las familias genéricas son:

- serif
- sans-serif
- monospace
- cursive
- fantasy

Las más interesantes
son las dos primeras.



Si queremos usar otro tipo de fuentes, nos enfrentamos al problema de la disponibilidad para el visitante.

Entonces, ¿cómo resolvemos esto?

Añadiendo fuentes web mediante CSS.

Fuentes Web

Fuentes de terceros Las fuentes web no siempre están preinstaladas en un dispositivo y, por lo tanto, deben ser descargadas en el navegador del usuario antes de ser mostradas. Algunos ejemplos de fuentes web incluyen la fuente Open Sans y Roboto de Google (<https://fonts.google.com/>) así como Acumin o Cortado de Adobe Fonts (<https://fonts.adobe.com/>).

CSS definiendo familias tipográficas

Independientemente del tipo de fuente que utilicemos y de cómo la integremos en nuestro sitio, la implementación de una tipografía es siempre igual

Podríamos escribir una línea como la siguiente:

```
css body { font-family: 'Open Sans', sans-serif; }
```

Una vez definida nuestra fuente, ya podemos aplicarle las propiedades que vimos en la clase anterior.

Podemos incluso trabajar con más de una tipografía dependiendo del diseño que se nos haya asignado.

Mientras que si se utilizan fuentes de terceros, como Google o Adobe Fonts, solo estamos incluyendo las fuentes mediante la vinculación a un activo externo. Alojar fuentes localmente significa que tenemos los archivos de fuentes en nuestro propio servidor. Para ello vamos a hacer uso de la directiva css @font-face. La regla @font-face permite cargar fuentes personalizadas en una página web. Una vez que se agrega a una hoja de estilo, la regla indica al navegador que descargue la fuente desde donde está alojada y luego la muestre como se especifica en el CSS.

@font-face rule

https://www.w3schools.com/cssref/css3_pr_font-face_rule.asp

Permite al autor especificar fuentes online para visualizar en sus páginas web. Al permitir a los autores proporcionar sus propias fuentes, @font-face elimina la necesidad de depender del número limitado de fuentes de usuarios instaladas en sus computadoras.

Iconos

¿Cómo podemos incluir iconos en nuestra web?

Al igual que como ocurre con las fonts web, es necesario, en primer lugar, seleccionar la librería de iconos que vamos a utilizar. Hay muchísimas librerías gratuitas y también de pago disponibles en la web. Estas son algunas de ellas:

- [Font Awesome](#)
- [Iconic](#)
- [Material Design](#)

Dependiendo de cual elijamos, son las instrucciones que deberemos seguir. Pero probemos con la librería Font Awesome.

Font Awesome es una librería como cualquier otra, por tanto, para usarla lo primero que debemos hacer es incluir esa librería —generalmente al inicio del documento—, o bien tener en nuestros servidor los archivos de fuente con que trabaja FontAwesome.

La forma más simple sería incluir la propia librería de ellos. Los pasos para hacerlo son:

1. Incluimos el enlace a nuestros iconos dentro del `<head>` del HTML:

```
<link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.15.2/css/all.css">
```

2. Insertar el ícono seleccionado en el lugar del `<body>` que más nos guste: `<i class="fas fa-exclamation-triangle"></i>`

Recordemos que también es posible tenerlos alojados en nuestro servidor. En ese caso, primero tenemos que descargarlos y, posteriormente, linkear en nuestro html el CSS y el JS correspondiente.



Importante: No olvidemos revisar la documentación. Ahí encontraremos todo lo necesario para una correcta implementación.

CLASE X

Modelo de cajas

El modelo de caja o box model hace referencia a la forma en que TODOS los elementos de HTML se comportan frente a la implementación de determinadas líneas de CSS.

Básicamente, lo que propone este modelo es que pensemos a cada elemento como una caja (de ahí su nombre) que posee cuatro costados (arriba, derecha, abajo, izquierda) y, que a su vez, puede ser afectado por diversas características como lo son:

- El ancho y el alto.
- El relleno entre sus costados y el contenido.
- El ancho, grosor y tipo de línea de sus bordes.
- El distanciamiento entre los demás elementos que lo rodean.

En este sentido, si llevamos a nuestro cerebro a pensar a cada etiqueta como un elemento que puede ser afectado por lo anterior, vamos a poder entender mucho mejor el flujo que cada elemento presenta respecto a sus elementos hermanos y respecto a su contexto en general.

1. Modelo de caja

En HTML todos los elementos se representan mediante cajas, eso se conoce como el modelo de caja. Cada caja se compone de contenido, relleno, bordes y márgenes.

Cada elemento en HTML es una caja, y esas cajas se componen de márgenes (margin), bordes (border), relleno (padding) y finalmente el contenido (content). Mediante CSS podemos manipular todas estas propiedades para cambiar la apariencia de cada elemento. A su vez, estas propiedades pueden aplicarse de manera diferente a los cuatro lados de cada caja (top, right, bottom y left). Muchas de las propiedades solo aplican a elementos de bloque o semibloque.



2.Width y height

Propiedad width

Si un elemento de bloque no tiene declarada la propiedad width, el ancho será igual al 100% de su padre contenedor. Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como porcentajes (%) o píxeles (px).

```
css      div {  
           width: 120px;  
         }
```

Propiedad height

Si un elemento no tiene declarado la propiedad height, el alto será igual a la altura que le proporcione su contenido. Sea un elemento de bloque o de línea. Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como pixeles (px). Para la altura de los elementos no se recomienda usar porcentaje.

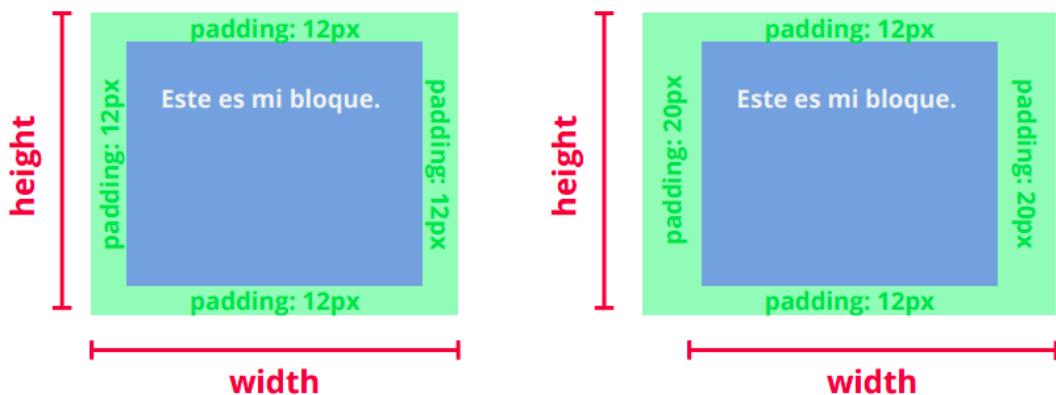
```
css      div {  
           height: 130px;  
         }
```

width y height en un elemento



3. Propiedad Padding

Es el espacio de relleno que podemos agregar entre el contenido del elemento y su borde.



Podemos asignarle un valor a esta propiedad usando cualquier unidad de medida, como píxeles (px), indicando 1 valor para los 4 lados de la caja.

```
css    div { padding: 12px; }
```

Único valor

Esto quiere decir que el relleno de **12px** se aplicará a todos los lados.

También podemos hacerlo con 2 valores. El primero va a indicar el padding de arriba y abajo, y el segundo el de la izquierda y la derecha.

```
css    div { padding: 12px 20px; }
```

Dos valores

12px de relleno para arriba y abajo.

20px de relleno para izquierda y derecha.

También podemos hacerlo con 3 valores. El primero va a indicar el padding de arriba, el segundo el de la izquierda y la derecha y el tercero será el de abajo.

```
css    div { padding: 12px 20px 18px; }
```

Tres valores

12px de relleno para arriba.

20px de relleno para izquierda y derecha.

18px de relleno para abajo.

Por último, podemos utilizar 4 valores que representarán los cuatro costados individualmente. Empezaremos por el valor de arriba y seguiremos (en el sentido de las agujas del reloj) por derecha, abajo e izquierda.

```
css   div { padding: 12px 15px 18px 21px; }
```

Cuatro valores

12px de relleno para arriba.
15px de relleno para la derecha.
18px de relleno para abajo.
21px de relleno para la izquierda.

4. Propiedades abreviadas

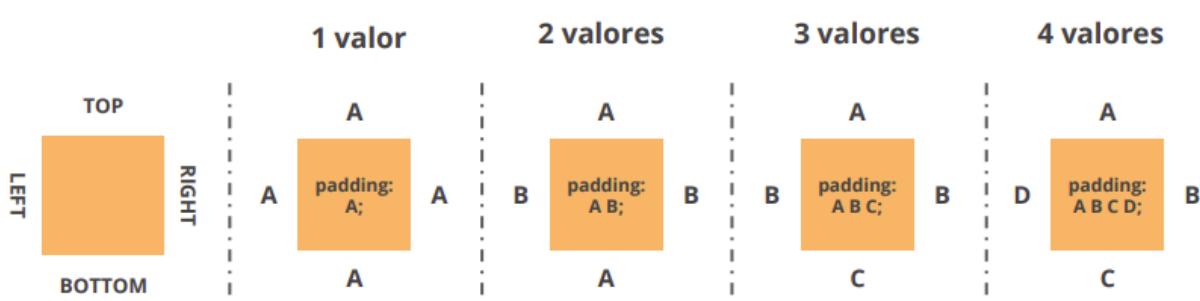
Como dijimos antes, muchas de las propiedades que vamos a ver se pueden aplicar de manera específica a uno de los costados de la caja. Cuando escribimos un solo valor para padding, por ejemplo:

```
css   div { padding: 15px; }
```

Lo que en realidad está pasando, sin que nos demos cuenta, es esto:

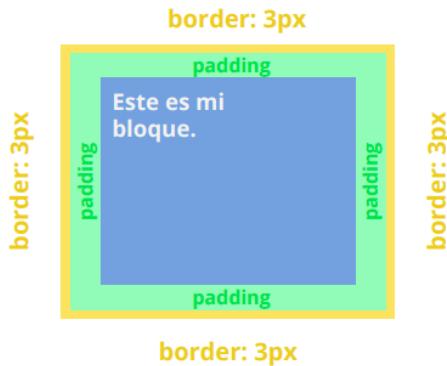
```
css   div {  
        padding-top: 15px; padding-right: 15px;  
        padding-bottom: 15px; padding-left: 15px;  
    }
```

En todas las propiedades de este estilo, como margin y padding, podemos escribir 1, 2, 3 o 4 valores, y según cuántos escribamos será a qué lados de la caja se apliquen los valores.



5.Propiedad Border

Hace referencia al borde del elemento. Se ubica entre el contenido y el margen.

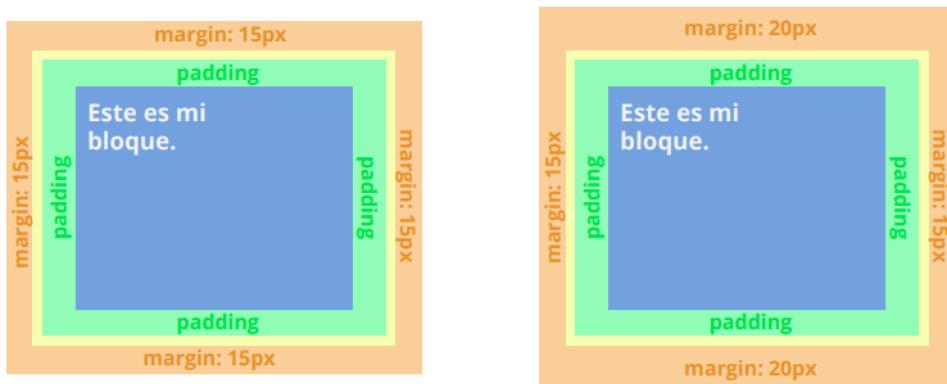


Podemos asignarle un valor a esta propiedad definiendo el estilo de línea, su espesor y su color. El estilo de línea puede ser solid, dotted, dashed o double. El espesor de línea puede ser cualquier unidad de medida de CSS. El color puede ser cualquier color válido de CSS.



6.Propiedad Margin

Hace referencia al margen exterior del elemento. Sirve para separar una caja de la otra.



Podemos asignarle valor a esta propiedad usando cualquier unidad de medida, como los píxeles (px), indicando 1 valor para los 4 lados de la caja.

```
css  div { margin: 15px }
```

Único valor

Esto quiere decir que el margen exterior de **15px** se aplicará a todos los lados: arriba, derecha, abajo e izquierda.

De la misma manera que con padding, podemos asignar también 2, 3 y hasta 4 valores para la propiedad.

```
css  div { margin: 20px 15px }
```

```
css  div { margin: 20px 15px 30px }
```

```
css  div { margin: 20px 15px 30px 25px }
```

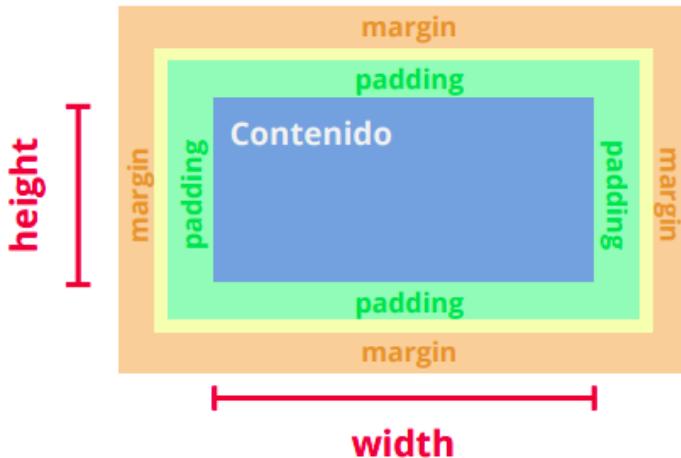
7.Box-sizing

Para poder calcular el ancho o el alto total de un elemento, tenemos que sumar todas las propiedades que vimos antes. Como se pueden imaginar, hacer este cálculo para saber cuánto va a ocupar un elemento finalmente es bastante trabajoso. La propiedad box-sizing nos ayuda con este cálculo.



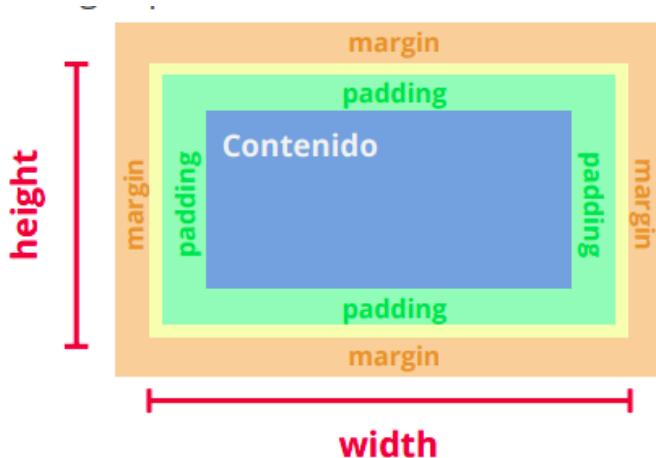
box-sizing: content-box

Por defecto su valor será content-box, y el comportamiento será el de aplicarle el ancho y alto que definamos al contenido del elemento.

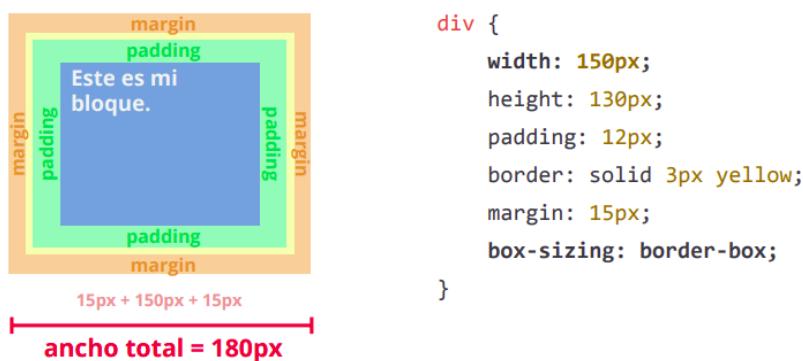


box-sizing: border-box

Si le asignamos el valor border-box, el ancho y alto que indiquemos tomará en cuenta no solo el contenido del elemento, sino también el padding y el borde, dejando solo el margen por fuera.



box-sizing: border-box



Es una práctica muy común asignarle box-sizing: border-box a todos los elementos del sitio con la siguiente línea de código: `* { box-sizing: border-box }`

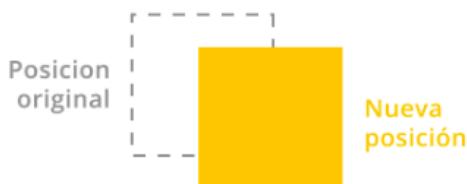
Position

Trabajar con elementos nos plantea algunas dudas, ¿qué pasa cuando necesitamos solapar un elemento por encima de otro?, ¿o cuando queremos que, por ejemplo, la barra de navegación de nuestro sitio web se quede inmóvil por más que hagamos scroll en el navegador? Este tipo de comportamientos son posibles en los otros tres tipos de posicionamiento que nos brinda CSS, los cuales buscan darnos más herramientas para lograr que nuestros documentos HTML tengan el mejor flujo posible dentro del navegador.

Introducción a posicionamiento

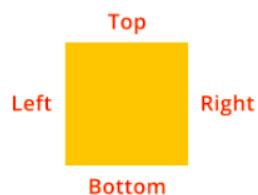
Trasladar elementos

El posicionamiento nos permite trasladar un elemento desde su posición original a una nueva posición. También nos permite superponer elementos



Puntos de referencia

Cada uno de los elementos de nuestra página web tiene cuatro puntos de referencia y esos son sus costados: superior, derecho, inferior e izquierdo. En CSS serán top, right, bottom y left.



Cuando desplazamos un elemento tomando un costado como referencia, empujaremos el elemento si el número es positivo o tiraremos de él si el número es negativo.



1. Posicionamiento relativo

El posicionamiento relativo nos permite trasladar un elemento desde su posición original a una nueva posición.

Cuando movemos una caja, el punto de referencia serán sus propios costados. Al posicionar la caja 1 de manera relativa, el espacio que ocupaba originalmente seguirá ocupado. Eso quiere decir que los elementos que estén a su lado (caja 2) no van a ser afectados por esta modificación.



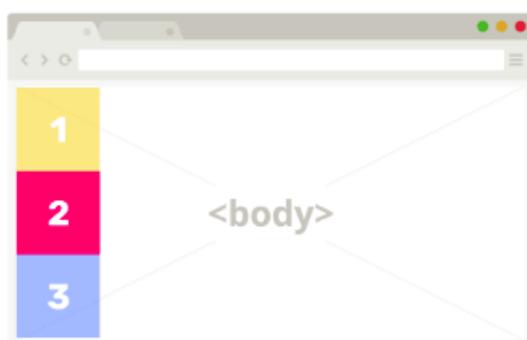
¿Cuándo se suele utilizar? Solemos utilizar posicionamiento relativo cuando queremos desplazar un elemento sin modificar el flujo original de los demás que están a su lado.

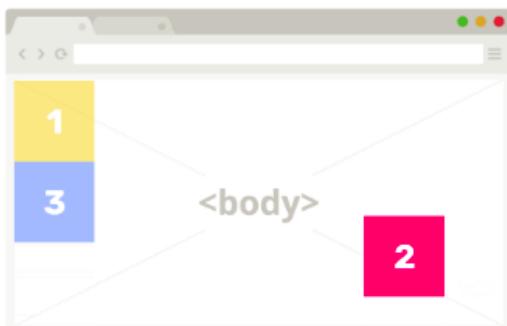


2. Posicionamiento absoluto

El posicionamiento absoluto nos permite trasladar un elemento tomando como referencia los costados del body.

Con el posicionamiento absoluto, los puntos de referencia serán los costados del body. Cuando movemos una caja de manera absoluta, el espacio que ocupaba quedará vacío y otros elementos podrán ocuparlo.

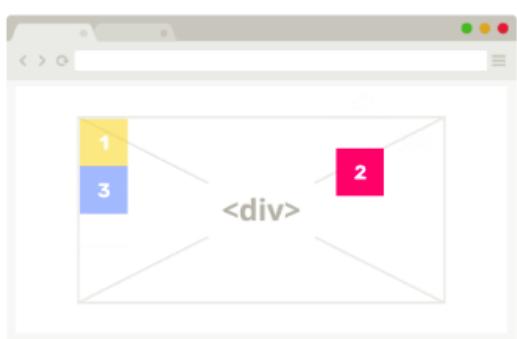




```
css .caja-2 { position: absolute; right: 100px; bottom: 50px; }
```

position relative + position absolute

Si nuestras cajas (hijas) están dentro de otra caja (padre), el punto de referencia seguirá siendo el body a menos que hagamos relativa la posición de su padre.



```
css div { position: relative } .caja-2 { position: absolute; right: 100px; top: 50px; }
```

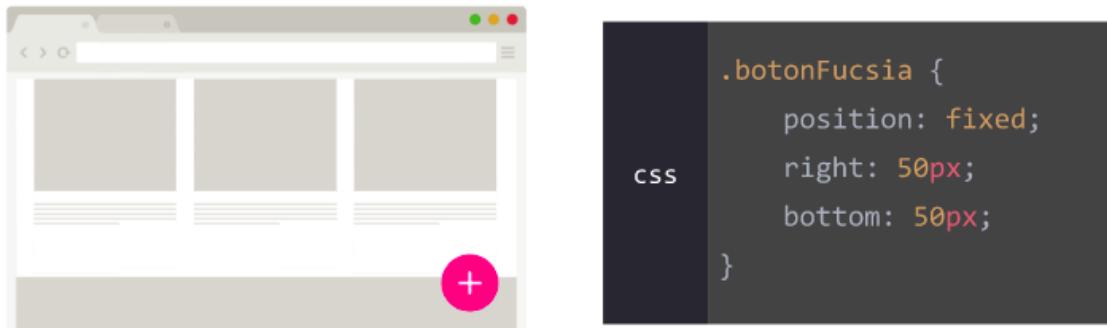
¿Cuándo se suele utilizar? Cuando queremos sacar un elemento del flujo normal y posicionarlo en un punto fijo con respecto a su contenedor o el body.



3. Posicionamiento fijo

El posicionamiento fijo nos permite trasladar un elemento tomando como referencia la ventana del navegador.

Con el posicionamiento fijo, los puntos de referencia serán los costados la ventana del navegador. Sin importar que hagamos scroll en la página, el elemento siempre se mantendrá fijo con respecto a la ventana del navegador.



¿Cuándo se suele utilizar? Cuando queremos que un elemento siga al usuario a medida que navega nuestro sitio. Por ejemplo, un botón fijo para siempre tener disponible la opción de crear un nuevo producto.

Posicionamiento Sticky

El posicionamiento sticky puede considerarse un híbrido de los posicionamientos relativo y fijo. Un elemento con posicionamiento sticky es tratado como un elemento posicionado relativamente hasta que cruza un umbral especificado, en cuyo punto se trata como fijo hasta que alcanza el límite de su parente.

¿Y cómo uso sticky?

Selector

Usamos el selector class para identificar el elemento.

css `.pegatina { position: sticky; top:15px; }`

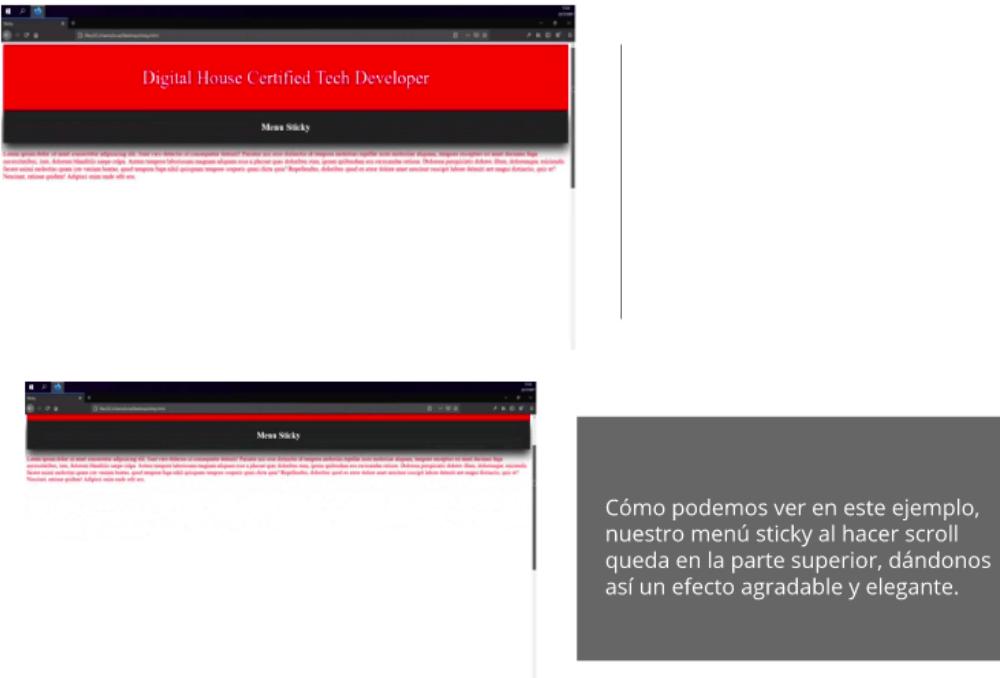
Valor

Como valor a las propiedades usamos, **sticky, 15px**.

Propiedades

En este ejemplo, hacemos uso de 2 propiedades CSS, la primera es **position** y, en este caso, usamos **top**. Pero puedes hacer uso de los siguientes valores, **top, left, right, bottom**

¿Qué pasa con el elemento si usamos sticky?



más info: https://www.w3schools.com/howto/howto_css_sticky_element.asp

Z-index

Seguramente hemos escuchado el concepto de "plano cartesiano", y lo primero que se nos viene a la cabeza son los cuadrantes y los ejes x e y.

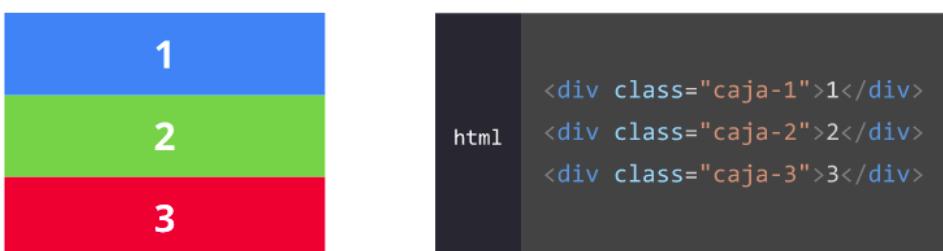
Y si bien esto no está del todo mal, hay un plano que quizás, por ahí, no estamos teniendo en cuenta... Nos referimos al eje z.

Pero ¿qué tiene que ver esto con HTML y CSS?

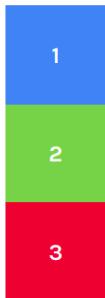
Pues bien, sucede que dentro de un documento HTML, y gracias a la magia de CSS, es posible llevar hacia atrás o traer hacia adelante un elemento determinado. En resumidas cuentas, lo estamos "moviendo" en el eje z, el de la profundidad.

El z-index nos permite cambiar el orden de las "capas" dentro de un documento HTML.

El z-index: Esta propiedad controla cómo se apilan las capas en CSS. Podemos considerar que cada elemento está en una capa diferente y que esas capas van en el orden en que aparecen los elementos en el HTML.

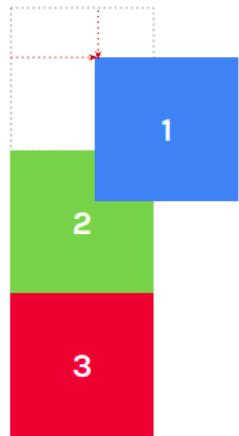


Solo podemos modificar el z-index de los elementos que tengan position: relative, absolute o fixed.



css

```
.caja-1 {  
    position: relative;  
}
```



Cualquier elemento al cual le asignemos position: **relative**, **absolute** o **fixed** se mostrará por encima del resto y tendrá un valor de **z-index** de **0**.

css

```
.caja-1 {  
    position: relative;  
    left: 100px;  
    top: 50px;  
}
```



Si a la segunda caja le agregamos la propiedad **z-index** con el valor **10**, se moverá delante del resto ya que ahora tiene un valor mayor.

css

```
.caja-1 { ... }  
.caja-2 {  
    position: relative;  
    z-index: 10;  
}
```

CLASE XI

Introducción a Flexbox

Flexbox es, sin lugar a dudas, una de las mejores y más recientes características del lenguaje CSS, pero ¿de qué trata?

Básicamente, Flexbox propone una manera de organizar los elementos presentes en nuestra estructura de HTML. Esta manera busca ser mucho más orgánica y fluida, pensada en los comportamientos naturales que posee cualquier elemento de la vida real con cuatros costados (arriba, derecha, abajo, izquierda).

Si bien hoy en día existen diversas maneras de lograr un diseño visual armónico y estético, Flexbox es una de las más elegidas. Su sencilla implementación la convierte en una excelente herramienta para reorganizar la estructura de nuestros documentos HTML.

Es una metodología de CSS que permite maquetar un sitio web utilizando una estructura de filas y columnas.

La historia de Flexbox

CSS (*Cascade Style Sheet*) nace en el año **1994**, y desde ese entonces fueron pocas las nuevas implementaciones que recibió el lenguaje.

Recién a mediados del **2008** se empieza a discutir la posibilidad de implementar una **nueva forma de maquetación**, que implique una **estructura de cajas flexibles**.

En el año **2011** salió a la luz el primer borrador con las especificaciones para implementar **Flexbox**. Al ser una **técnica nueva**, los navegadores en su versión comercial todavía no le daban suficiente **sopporte**.



Ventajas de Flexbox

Cuando usamos flotación para posicionar un elemento en un sitio web, el mismo deja de formar parte del flujo natural de la estructura de elementos. Esto genera solapamiento de cajas y estructuras difíciles de mantener. Flexbox propone un único flujo, en el que dispondremos de los elementos con mayor libertad para distribuir, redimensionar y reordenar cada uno de ellos en función de ese flujo de trabajo.

Ejes

¿Qué es un eje?

Si tuviéramos que definir esto en nuestra vida real, podríamos decir que un eje es aquella representación imaginaria a partir de líneas que definen la dirección frente a la cual se desplaza un elemento.

Con esto en mente, podemos decir, entonces, que los ejes dentro de un contenedor flex definen la orientación a partir de la cual se desplazarán los elementos internos del mismo.

Un contenedor flex posee dos ejes (el X y el Y): el eje principal, llamado también **main axis**, y el eje transversal, llamado **cross axis**.

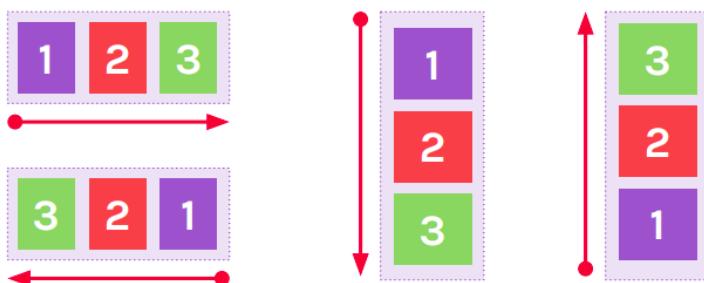
Algo particular dentro de este tipo de posicionamiento es que tanto el main axis y el cross axis no están definidos ni por el plano horizontal ni por el plano vertical. Si no que todo dependerá de la configuración que nosotros dispongamos al momento de escribir nuestro código.

Cuando trabajamos en un flujo flex, hablamos del main axis y el cross axis. Definiendo el eje principal de nuestro contenedor flex estamos determinando el flujo que tendrán los elementos dentro del contenedor. En función de cuál es el eje principal, los elementos se distribuyen en filas horizontales o en columnas verticales.



flex-direction

Con esta propiedad definimos el main axis (eje principal) del contenedor, que puede ser tanto horizontal como vertical. El cross axis (eje transversal) será la dirección perpendicular al main axis.



flex-direction: row

Los ítems se disponen en el **eje x, de izquierda a derecha**. Si no le aclaramos la propiedad `flex-direction` al contenedor, `row` es el valor por defecto.



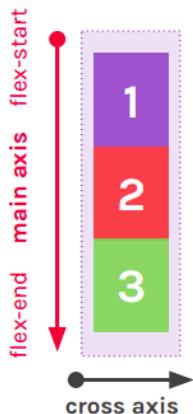
`flex-direction: row-reverse`

Los ítems se disponen en el eje x, de derecha a izquierda. En este caso, estamos **invirtiendo el inicio y fin del main-axis**.



`flex-direction: column`

Los ítems se disponen en el eje y, de arriba hacia abajo.



`flex-direction: column-reverse`

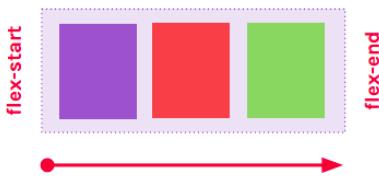
Los ítems se disponen en el eje y, de abajo hacia arriba. En este caso, estamos **invirtiendo el inicio y fin del main-axis**.



Flexbox nos da dos propiedades para alinear fácilmente los elementos. A través del main axis con justify-content. A través del cross axis con align-items.

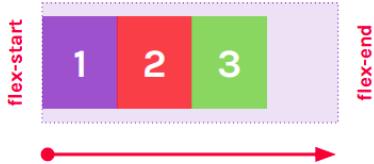
justify-content

Con esta propiedad **alineamos los ítems a lo largo del main axis**. Si es **horizontal**, se alinearán en función de la **fila**. Si es **vertical**, se alinearán en función de la **columna**.



justify-content: flex-start

Los ítems se alinean respecto del **inicio del main axis** que hayamos definido. Si no le aclaramos el justify-content al contenedor, **flex-start es el valor por defecto**.



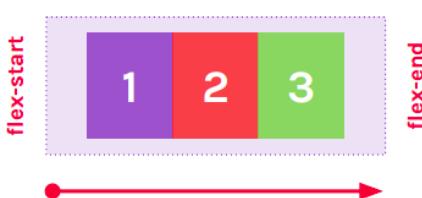
justify-content: flex-end

Los ítems se alinean respecto del **final del main axis** que hayamos definido.



justify-content: center

Los ítems se alinean en el **centro del main axis**.



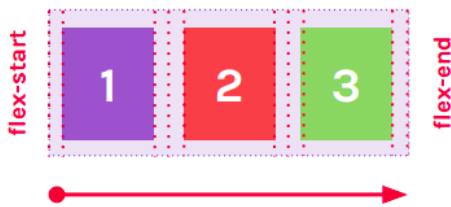
justify-content: space-between

Los ítems se distribuyen de manera uniforme. El **primer ítem será enviado al inicio** del main axis, y el **último ítem, al final**. El **espacio libre se repartirá** para separar los ítems.



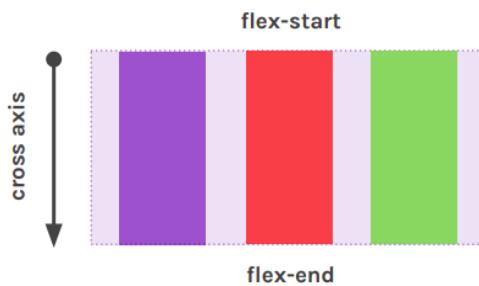
justify-content: space-around

Los ítems se distribuyen de **manera uniforme**. El espacio libre disponible se repartirá entre todos los elementos. Del espacio que le toque a cada elemento, la mitad irá a la derecha y la otra a la izquierda (o arriba y abajo en caso de que sean columnas).



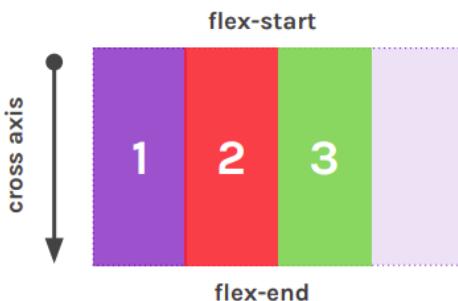
align-items

Con esta propiedad alineamos los ítems a lo largo del cross axis. Si no aclaramos esta propiedad, el valor por defecto es **stretch**, en otras palabras, los ítems ocuparán todo el espacio disponible en el cross axis.



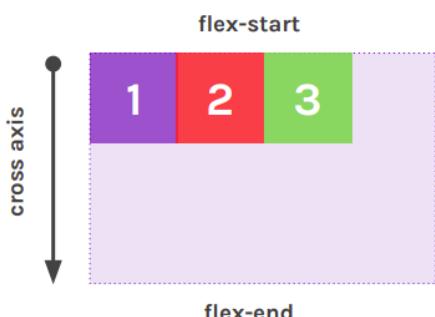
align-items: stretch

Los ítems se ajustan para **abrir todo el contenedor**. Si el cross axis es vertical, se ajustan en función de la columna. Si el cross axis es horizontal, se ajustan en función de la fila.



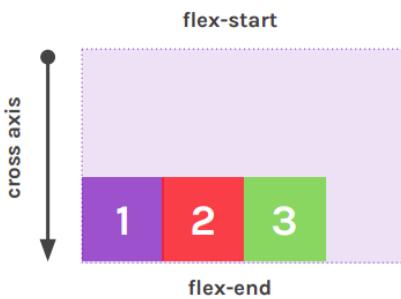
align-items: flex-start

Los ítems se alinean al **inicio** del cross-axis.



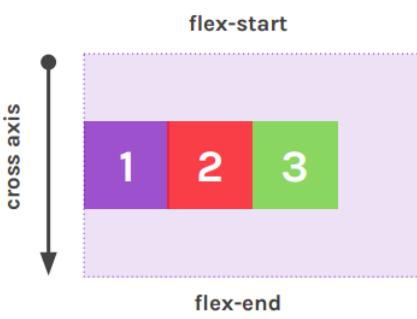
align-items: flex-end

Los ítems se alinean al **final** del eje transversal.



align-items: center

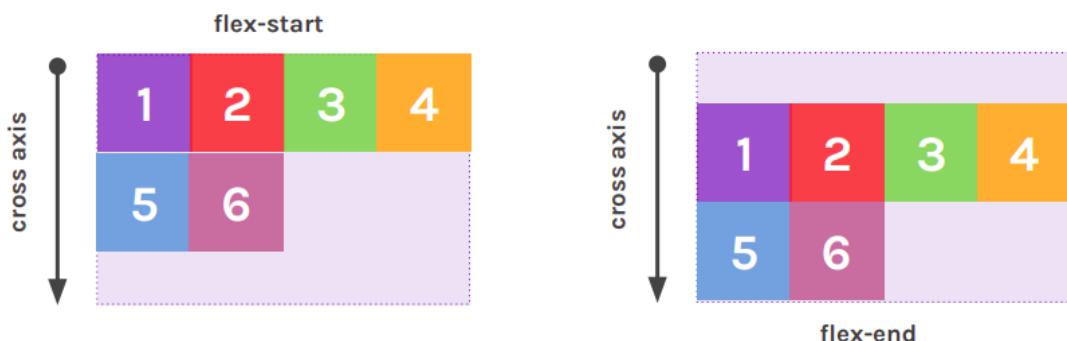
Los ítems se alinean al **centro** del eje transversal.



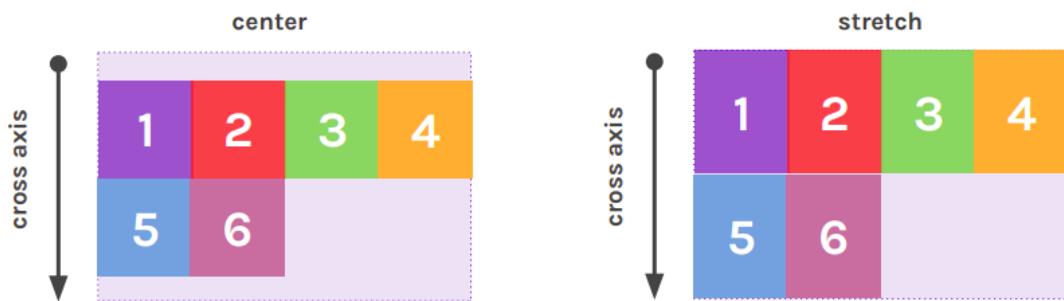
align-content

Si tenemos un contenedor de una sola línea (donde flex-flow se establece como no-wrap) utilizaremos align-items, pero **en el caso de que estemos trabajando con contenedores multilínea debemos utilizar align-content**. Con esta propiedad **alineamos los ítems a lo largo del cross axis cuando los contenedores flexibles incluyen de varias líneas** (donde flex-flow se establece en wrap o wrap-reverse). Los valores que admite la propiedad align-content son similares a los que podemos utilizar para align-items. Vale la pena probar como funcionan ambas propiedades para entenderlas bien.

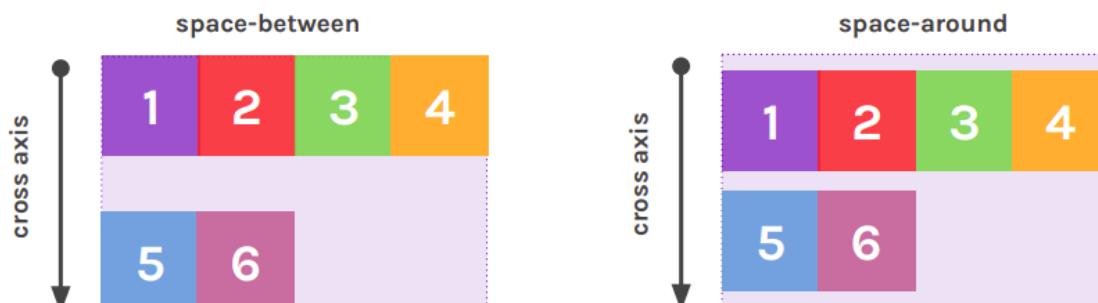
align-content: flex-start | flex-end



align-content: center | stretch



align-content: space-between | space-around



Estructura básica

El posicionamiento de tipo flex posee una serie de configuraciones básicas que nos van a permitir trabajar de una manera mucho más cómoda y práctica al momento de escribir nuestras hojas de estilo CSS.

La estructura básica de un contenedor flex no es otra cosa más que todas aquellas líneas de CSS que, sí o sí, deben estar presentes para que el diseño y estética visual buscado se pueda implementar de la mejor manera posible.

Flexbox propone una estructura basada en el uso de un contenedor **padre (flex-container)** y sus elementos **hijos (flex-items)**.

Trabajar con Flexbox

Para empezar a trabajar con Flexbox tenemos que definir un flex-container. Para eso usamos la propiedad **display con el valor flex**. De esta forma, estamos habilitando un contexto flex, para trabajar con los hijos directos del elemento. La propiedad display también puede recibir el valor inline-flex.

```
css .contenedor-padre { display: flex; }
```

Estructura básica

Cuando hablamos de un flex-container, hablamos de un elemento HTML que contiene a uno o más elementos. A estos elementos anidados los llamamos flex-items. En el flex-container es en donde configuramos la mayoría de las propiedades flex.



flex-wrap

Por defecto, los elementos hijos de un contenedor flex van a tratar de entrar todos en una misma línea.



Para aclararle al contenedor que debe **respetar el ancho definido de sus hijos** usamos la propiedad **flex-wrap** con el valor **wrap**.



Como dijimos, la propiedad **flex-wrap con el valor wrap** permitirá que los ítems tomen el ancho definido y que los que no entren en la línea, caigan a la siguiente. **flex-wrap** también puede recibir los valores **nowrap** y **wrap-reverse**.

```
css .contenedor-padre {  
    display: flex;  
    flex-wrap: wrap;  
}
```

flex-items

Un flex-item, a su vez, puede convertirse en un flex-container. Para eso, solo hace falta asignarle la regla **display:flex**, para que así sus elementos hijos pasen a ser flex-items.

```
css .elemento-hijo {  
    display: flex;  
}
```

Items

Cuando trabajamos con el posicionamiento Flexbox, generalmente todo lo implementamos directamente en el elemento padre contenedor.

Pero ¿qué pasa si deseamos modificar el comportamiento de un flex item?

order

Con esta propiedad controlamos el orden de cada ítem, sin importar el orden original que tengan en la estructura HTML. Esta propiedad recibe un número entero, positivo o negativo, como valor. Por defecto, todos los ítems flex tienen un order: 0 implícito, aunque no se especifique.

```
css .caja { order: 1; }
```

order: número positivo

Si le asignamos a la caja Q (que posee la clase caja-q) la propiedad order con valor 1, esta pasará al final de la fila por ser el número más alto. Recordemos que, por defecto, el valor del orden de cada ítem es 0.



order: número negativo

Si ahora le asignamos a la caja D la propiedad order con un -1 como valor, esta pasará al principio de la fila. Colocando al ítem con el orden más pequeño primero.



flex-grow

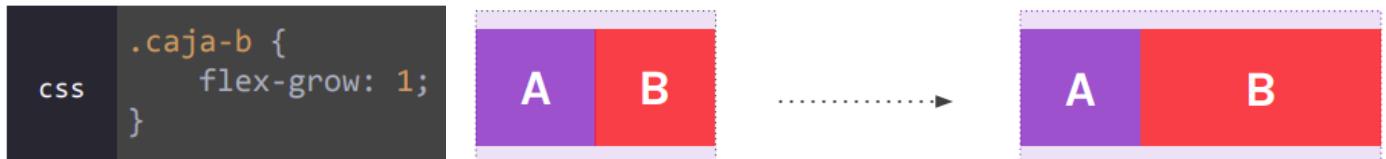
Con esta propiedad definimos cuánto puede llegar a crecer un ítem en caso de disponer de espacio libre en el contenedor. Configura un crecimiento flexible para el elemento.



Si **ambos ítems** tienen la propiedad flex-grow con valor 1, a medida que el contenedor se agrande, irán abarcando el espacio disponible en partes iguales.



Si **un solo ítem** tienen la propiedad flex-grow, este intentará ocupar el espacio libre disponible, a medida que el contenedor se agrande, según la proporción que definamos con el valor.

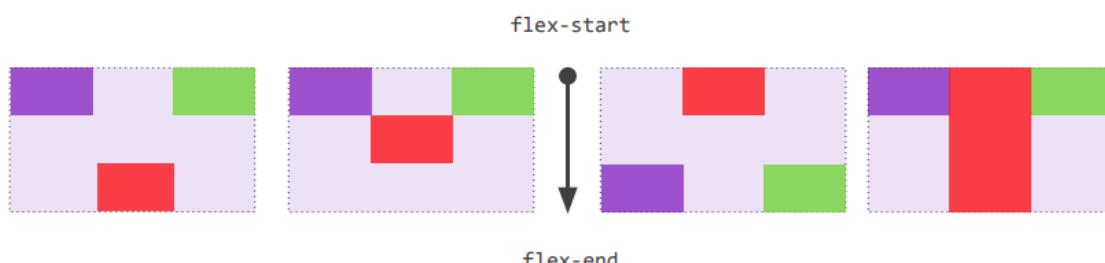


El número que le asignamos a flex-grow determina qué cantidad de espacio disponible dentro del contenedor flexible tiene que ocupar ese ítem. **1 equivale al 100% del espacio disponible, y 0 al 0%.** Podemos usar cualquier valor en el medio, **como 0.25 para el 25%.**



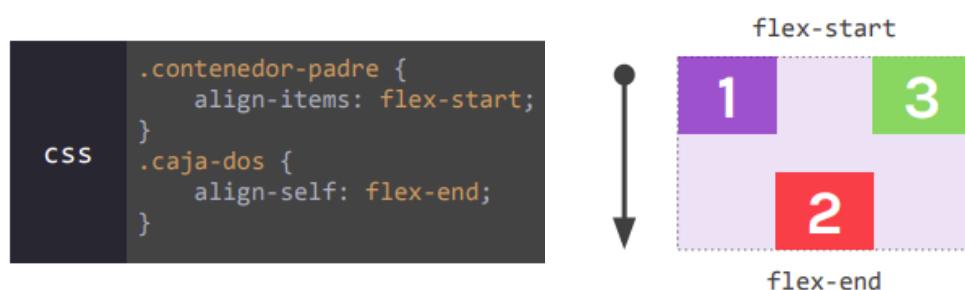
align-self

Nos permite alinear, sobre el cross axis, a cada ítem al que le apliquemos esta propiedad, independientemente de la alineación que se haya definido en el contenedor flex con align-items.



align-self: flex-end

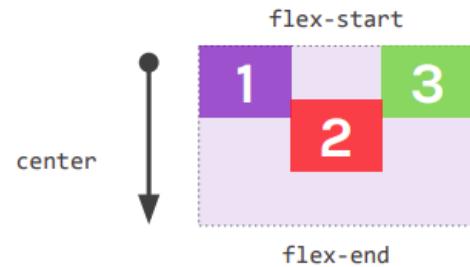
Con flex-end, el ítem se alinea al final del eje transversal.



align-self: center

Con center, el ítem se alinea al centro del eje transversal.

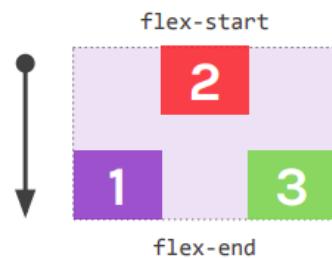
```
css   .contenedor-padre { align-items: flex-start; }
      .caja-dos { align-self: center; }
```



align-self: flex-start

Con flex-start el ítem se alinea al inicio del eje transversal.

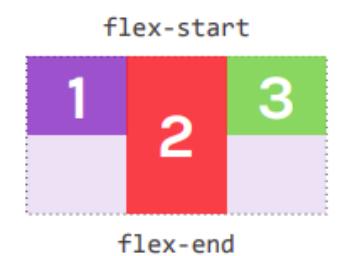
```
css   .contenedor-padre { align-items: flex-end; }
      .caja-dos { align-self: flex-start; }
```



align-self: stretch

Con stretch, el ítem se ajusta hasta abarcar todo el cross axis, es el comportamiento por defecto. Funciona siempre que el elemento no tenga definida una altura.

```
css   .caja-uno, .caja-tres { align-self: flex-start; }
      .caja-dos { align-self: stretch; }
```



Estas propiedades aplicarán para los flex-items siempre y cuando el contenedor padre sea un flex-container.