Laboratorio de Computación II

Unidad 3: Hojas de Estilo en Cascada (CSS)

Tecnicatura Universitaria en Programación - UTN

CSS - Introducción

- CSS son las siglas en inglés de Cascading Style Sheets (Hojas de Estilo en Cascada).
- Es el lenguaje que usamos para diseñar un documento HTML.
- Describe cómo se deben mostrar los elementos HTML.
- CSS ahorra mucho trabajo. Puede controlar el diseño de varias páginas web a la vez

CSS - Maneras de insertar código CSS a HTML

CSS Externo:

En el HTML debemos incluir una referencia al archivo de hoja de estilo (**.css**) dentro del elemento **<link>**.

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/index.css">
</head>
```

css/**index.css**

```css
h1 {
    color: red;
    text-align: center;
}
```

CSS Interno:

Se define dentro de la etiqueta **<style>** de la página.

```html
<!DOCTYPE html>
<html>
<head>
    <style>
        h1 {
            color: red;
            text-align: center;
        }
    </style>
</head>
```

CSS en Línea:

Se define dentro de la propiedad **STYLE** de un elemento por lo que el estilo aplicará sólo al elemento en cuestión.

```html
<h1 style="color: red; text-align: center;">
    Título principal
</h1>
```

RECOMENDACIÓN: agregar CSS Externo e Interno dentro del elemento **<head></head>**.

CSS - Comentarios

Al igual que en HTML, en CSS podemos definir comentarios en nuestro código que nos ayuden a enmarcar el estilo de un elemento o que nos permitan describir qué se está haciendo programáticamente. Los comentarios en CSS se debe colocar entre /* y */

```
/* Este es
    un comentario
*/
h1 {
    color: red; /* Este es otro comentario*/
    text-align: center;
}
```

CSS - Sintaxis

Al escribir código CSS vamos a definir **REGLAS**. Cada una de estas reglas está compuesta por un **SELECTOR** y un **BLOQUE DE DECLARACIÓN**, donde en el bloque de declaración se colocarán las propias **DECLARACIONES** compuestas por pares **PROPIEDAD - VALOR**, cada una de ellas separadas por punto y coma (**;**).

CSS - Selectores

Se utilizan para “seleccionar” los elementos HTML que queremos diseñar

Selectores de Elementos:

Selecciona elementos HTML en función de su propio nombre.

```css
p {
    color: gray;
    font-size: 16px;
}
```

Selector de ID (#):

Selecciona el elemento HTML que coincide con el identificador (atributo “id”) del elemento HTML.

```html
<img id="img-coding" src="img/coding.png" alt="Persona codeando">
```

```css
#img-coding {
    width: 100px; height: auto;
}
```

Selector de Clase (.):

Selecciona todos los elementos HTML que coinciden con el valor de la clase (atributo “class”) de los elementos HTML.

```html
<h1 class="text-center">Título centrado</h1>
<p class="text-center background-gray">Texto centrado</p>
```

```css
.text-center {
    text-align: center;
}
.background-gray {
    background-color: #dcdcdc;
}
```

NOTA: no es posible tener dos elementos HTML en un mismo documento con el mismo valor para el atributo ID pero sí es posible tener múltiples elementos con una misma clase y al mismo tiempo, un elemento puede tener múltiples clases.

CSS - Selectores

Selector Universal (*):

Selecciona a todos los elementos html del documento.

```css
* {
    text-align: center;
    color: blue;
}
```

Selector de Agrupación:

Con esta característica de CSS podremos darle el mismo estilo a múltiples selectores definiendo una única regla.

Mire el siguiente código CSS (los elementos **h1**, **h2** y **p** tienen las mismas definiciones de estilo):

```css
h1 {
    text-align: center;
    color: red;
}

h2 {
    text-align: center;
    color: red;
}

p {
    text-align: center;
    color: red;
}
```

Es mejor agrupar los selectores para reducir la líneas de código, separando los selectores con una coma (,) como se puede ver a continuación:

```css
h1, h2, p {
  text-align: center;
  color: red;
}
```

CSS - Selectores

Selector Descendiente:

En caso de querer dar estilo a un elemento en particular que se encuentra dentro de otro elemento, lo podemos hacer escribiendo cada uno de los elementos/id/clase según la anidación que tienen en HTML separados por un espacio.

```
<style>
article h3 {
    color: red;
}
</style>
<h3>Listado de artículos</h3>
<article>
    <h3>Artículo #1</h3>
</article>
<article>
    <h3>Artículo #2</h3>
</article>
```

Selector “>” o «mayor qué» (selector de hijos):

El selector «mayor qué» es utilizado en CSS para seleccionar todos los elementos que sean directamente descendientes de otro, es decir, que sean hijos directos de un determinado elemento padre, por eso también se conoce como “selector de hijos”.

```
<style>
div > a {
    color: red;
}
</style>
<div>
    <p><a href="#">Enlace 1</a></p>
    <p><a href="#">Enlace 2</a></p>
    <a href="#">Enlace 3</a>
    <a href="#">Enlace 4</a>
</div>
```

CSS - Múltiples estilos

Si se han definido algunas propiedades para el mismo selector (elemento) en diferentes hojas de estilo, se utilizará el valor de la última hoja de estilo leída.
Veamos un ejemplo:

index.css

```css
p {
  text-align: center;
  color: red;
}
```

page-about.css

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

page-about.html

```html
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="css/index.css">
    <link rel="stylesheet" href="css/page-about.css">
</head>
<body>
    <p>Texto de ejemplo</p>
</body>
</html>
```

El resultado que obtendremos es el siguiente:

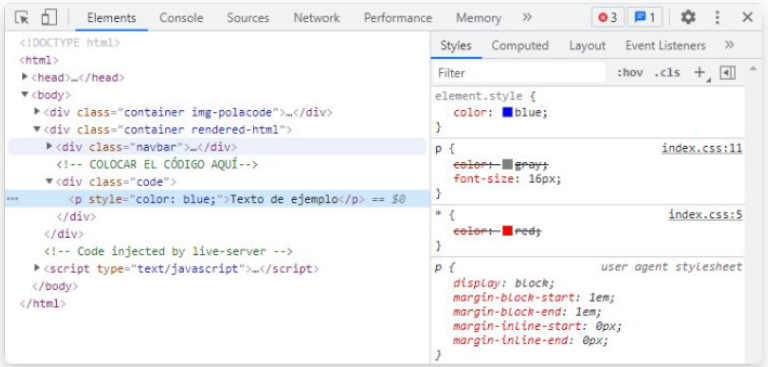
CSS - Especificidad

Si hay dos o más reglas CSS en conflicto que apuntan al mismo elemento, el navegador sigue algunas reglas para determinar cuál es la más específica y, por lo tanto, aplicarla al elemento en cuestión.

Debemos pensar a la **especificidad** como una **puntuación** que determina qué declaraciones de estilo se aplican en última instancia a un elemento.

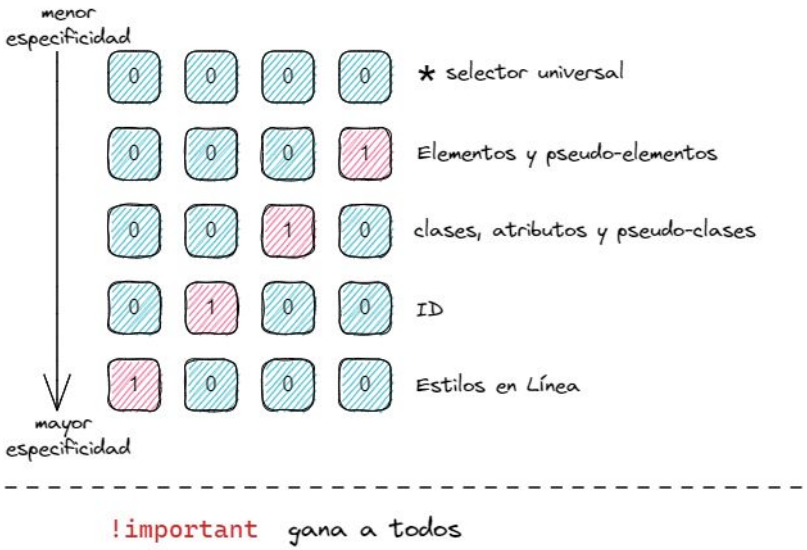
El selector universal (*) tiene baja especificidad, mientras que los selectores de ID son muy específicos.

NOTA: la especificidad es una razón común por la que las reglas CSS no se aplican a algunos elementos, aunque cree que deberían hacerlo.



CSS - Especificidad - Jerarquía

Cada selector tiene su lugar en la jerarquía de especificidad. Hay cuatro categorías el cual define el nivel especificidad de un selector:



Si tenemos dos hojas de estilo en las que para un elemento se define la misma propiedad y las cuales contienen la misma especificidad, el navegador aplicará el estilo de la última hoja de estilo agregada.

```
<head>
    <link rel="stylesheet" href="css/index.css">
    <link rel="stylesheet" href="css/page-about.css">
</head>
```

En el ejemplo, **page-about.css** tiene un mayor “peso” o jerarquía porque se ha agregado luego de **index.css**

Más info: https://developer.mozilla.org/es/docs/Web/CSS/Specificity
https://dev.to/lupitacode/especificidad-en-css-que-es-y-como-funciona-52k6

CSS - !important

La regla **!important** en CSS se usa para agregar más importancia a una propiedad/valor de lo normal.
De hecho, al usar la regla **!important**, anulará TODAS las reglas de estilo anteriores para esa propiedad específica en ese elemento.
Veamos un ejemplo:

```
p {
  background-color: red !important;
}

p#my-id {
  background-color: blue;
}

p.my-class {
  background-color: gray;
}
```

En el ejemplo anterior. los párrafos obtendrán un color de fondo rojo, aunque el selector de ID y el selector de clase tienen una mayor especificidad, la regla **!important** anula la propiedad background-color en ambos casos.

SUGERENCIA: es bueno conocer la regla !Important, es posible que la vea en algún código fuente CSS. Sin embargo, no se recomienda usarla a menos que sea absolutamente necesario.

CSS - Box Model (Modelo de Caja)

Todos los elementos HTML pueden ser considerados como **CAJAS**.

El término "modelo de caja" se utiliza cuando se habla de diseño y maquetación.

El modelo de caja CSS es esencialmente una caja que envuelve cada elemento HTML. Consiste en: márgenes, bordes, relleno y el contenido real (en la imagen ilustra el modelo de caja)

- **content**: el contenido del cuadro, donde aparecen el texto y las imágenes.
- **padding**: limpia un área alrededor del contenido. El acolchado es transparente
- **border**: un borde que rodea el relleno y el contenido.
- **margin**: deja un espacio fuera del borde. El margen es transparente

CSS - Margin

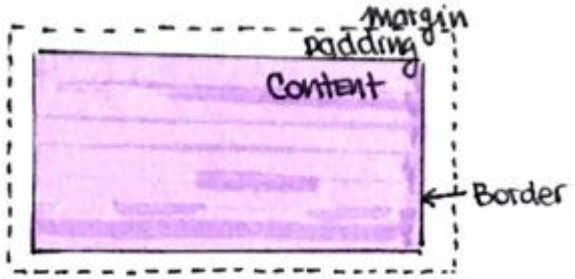
Se utiliza para crear espacio alrededor de los elementos, fuera de los bordes definidos. En otras palabras, es la propiedad que nos permite definir cuán alejado estará un elemento de los otros que lo rodean.

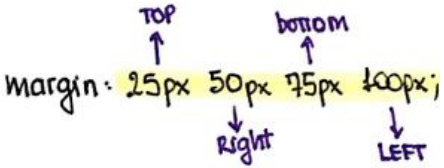
CSS tiene **propiedades** para especificar el margen de cada lado del elemento:

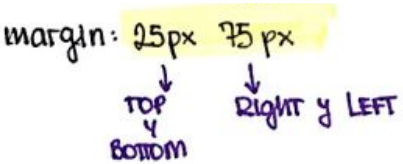
- **margin-top**: espacio hacia arriba
- **margin-right**: espacio hacia la derecha
- **margin-bottom**: espacio hacia abajo
- **margin-left**: espacio hacia la izquierda

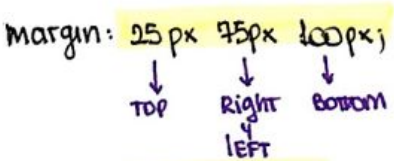
Esas propiedades pueden contener los siguientes valores:

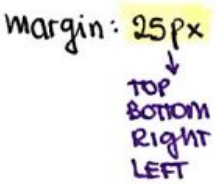
- **auto**: el navegador calcula el margin.
- **longitud**: en px, em, etc. (ej: 20px)
- **porcentaje**: especifica un margen en % del ancho del elemento contenedor. (ej: 10%)











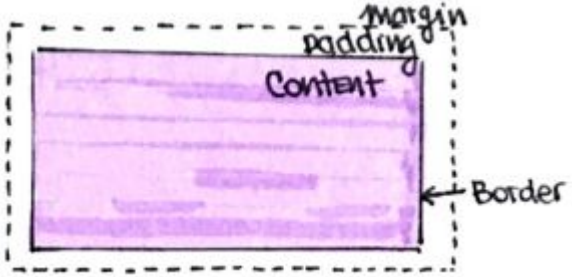
CSS - Padding

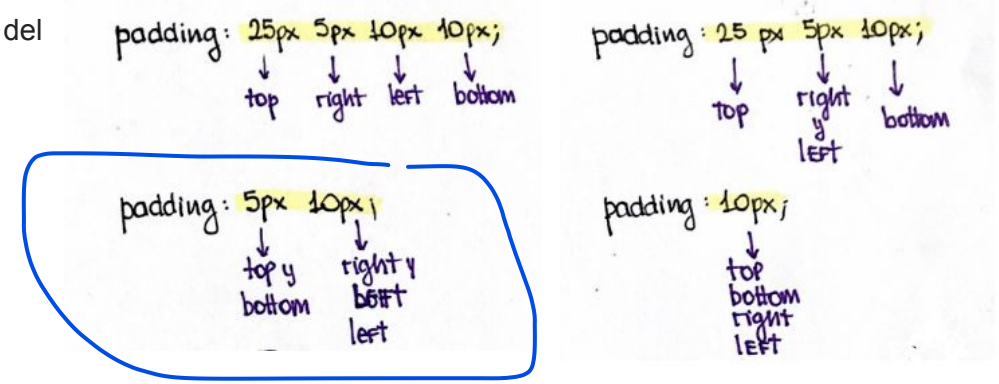
Se utiliza para generar espacio alrededor del contenido de un elemento dentro de los bordes definidos.

- **padding-top**: espacio hacia arriba
- **padding-right**: espacio hacia la derecha
- **padding-bottom**: espacio hacia abajo
- **padding-left**: espacio hacia la izquierda

Esas propiedades pueden contener los siguientes valores:

- **longitud**: en px, em, etc. (ej: 20px)
- **porcentaje**: especifica un margen en % del ancho del elemento contenedor. (ej: 10%)





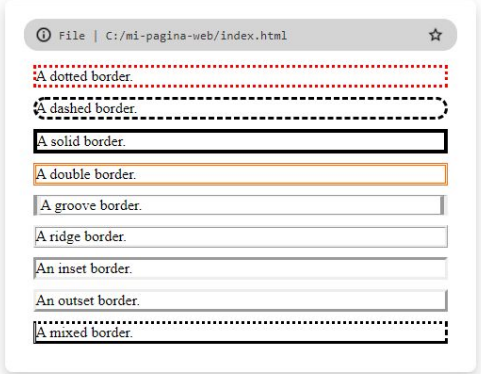
CSS - Border (border)

Permiten especificar el estilo, el ancho y el color del borde de un elemento (border, border-top, border-right,border-bottom, border-left).

Estas son las principales propiedades relacionadas al borde:

- **border-color**: color del borde
- **border-style**: estilo del borde (dashed, solid, dotted, double, groove, inset, outset, none, hidden)
- **border-width**: ancho del borde, pudiendo setear el mismo valor para los 4 bordes o uno para cada uno (0px, 4px, 8px, 12px) (top, right, bottom y left respectivamente)
- **border-radius**: nos permite redondear las puntas de la caja.

```
<style>
  p { margin: 10px }
    p.dotted {border-style: dotted; border-color: red}
    p.dashed {border-style: dashed; border-radius: 30px}
    p.solid {border-style: solid; border-width: 4px}
    p.double {border-style: double; border-color: #ff6600}
    p.groove {border-style: groove; border-width: 2px 8px}
    p.ridge {border-style: ridge;}
    p.inset {border-style: inset;}
    p.outset {border-style: outset;}
    p.mix {border-style: dotted dashed solid double;}
</style>
<p class="dotted">A dotted border.</p>
<p class="dashed">A dashed border.</p>
<p class="solid">A solid border.</p>
<p class="double">A double border.</p>
<p class="groove">A groove border.</p>
<p class="ridge">A ridge border.</p>
<p class="inset">An inset border.</p>
<p class="outset">An outset border.</p>
<p class="mix">A mixed border.</p>
```



CSS - Height y Width (alto y ancho)

Las propiedades de **height** y **width** se utilizan para establecer el alto y el ancho de un elemento.

Los **valores** que pueden tomar estas propiedades son:

- **auto**: es el valor predeterminado. El navegador calcula la altura y el ancho
- **length (longitud)**: define la altura/ancho en px, cm, etc.
- **% (porcentaje)**: define la altura/ancho en porcentaje del bloque contenedor (ej: 50%)
- **initial (inicial)**: establece la altura/ancho en su valor predeterminado
- **inherit (heredar)**: el alto / ancho se heredará el valor de su elemento padre.

Contamos con las propiedades **max-height** / **min-height** para establecer alto máximo y mínimo, como así también contamos con **max-width** / **min-width** para el ancho del elemento.

```
<style>
    div {
        margin: 10px 0px;
    }
    .div1 {
        width: 100px;
        height: 50px;
        border: 2px solid blue;
    }
    .div2 {
        width: 50%;
        height: 50px;
        border: 2px solid red;
    }
</style>
<div class="div1"></div>
<div class="div2"></div>
```

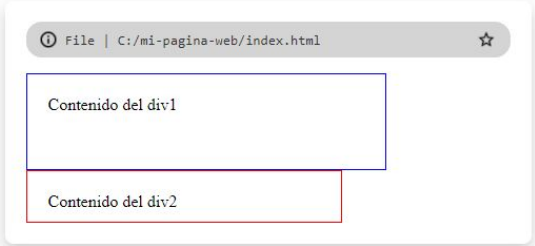
CSS - box-sizing

Esta propiedad (**box-sizing**) indica cómo se debe calcular el tamaño de un elemento (alto, y ancho), en otras palabras, nos nos permite incluir o no el **padding** y el **border** en el ancho y alto total de un elemento.

Valores:

- **content-box**: (por defecto) las propiedades width y height no incluyen el border ni el padding.
- **border-box:** el border y el padding se incluyen dentro del width y el height

```
<style>
    .div1 {
        width: 300px;
        height: 50px;
        padding: 20px;
        border: 1px solid blue;
        box-sizing: content-box;
    }
    .div2 {
        width: 300px;
        height: 50px;
        padding: 20px;
        border: 1px solid red;
        box-sizing: border-box;
    }
</style>
<div class="div1">
    Contenido del div1
</div>
<div class="div2">
    Contenido del div2
</div>
```




```
* {
    box-sizing: border-box;
}
```

De esta manera seteamos el box-sizing para asegurarnos que los elementos no ocupen más espacio del que le seteamos con width y height.

¿Preguntas?

CSS - Unidades y medidas - Absolutas

Las medidas absolutas son fijas en relación con otras. Estas consisten en medidas físicas conocidas y los píxeles.

Unidad	Nombre	Equivale a
cm	Centímetros	1cm = 96px/2,54
mm	Milímetros	1mm = 1/10 de 1cm
Q	Cuartos de milímetros	1Q = 1/40 de 1cm
in	Pulgadas	1in = 2,54cm = 96px
pc	Picas	1pc = 1/16 de 1in
pt	Puntos	1pt = 1/72 de 1in
px	**Pixeles**	**1px = 1/96 de 1in**

CSS - Unidades y medidas - Relativas

Las unidades de longitud relativa son relativas a algo más, por ejemplo, al tamaño de letra del elemento principal o al tamaño de la ventana gráfica. La ventaja de usar unidades relativas es que con una planificación cuidadosa puedes lograr que el tamaño del texto u otros elementos escalen en relación con todo lo demás en la página.

Unidad	Relativa a
em	Tamaño de letra del elemento padre, en el caso de propiedades tipográficas como font-size, y tamaño de la fuente del propio elemento en el caso de otras propiedades, como width.
rem	Tamaño de la letra del elemento raíz (elemento html).
vw	1% del ancho de la ventana gráfica.
vh	1% de la altura de la ventana gráfica.
vmin	1% de la dimensión más pequeña de la ventana gráfica.
vmax	1% de la dimensión más grande de la ventana gráfica.
%	Porcentaje en relación del valor de la misma propiedad en el elemento padre.

CSS - Texto

Como vimos anteriormente, el texto incluido en un elemento se dispone dentro de la **caja de contenido del elemento**. Esta empieza en la parte superior izquierda del área de contenido y fluye hacia el final de la línea. Una vez que llega al final, baja a la línea siguiente y sigue, y luego continúa a la línea siguiente, hasta que todo el contenido se ha ubicado en la caja. El contenido de texto se comporta efectivamente como una serie de elementos en línea, distribuidos en líneas adyacentes entre sí, y sin crear saltos de línea hasta que se llega al final de la línea, a menos que se fuerce un salto de línea manual con el elemento
.

Las propiedades CSS que se usan para aplicar estilo al texto pueden clasificarse generalmente en dos categorías:

- **Estilos del tipo de letra**
- **Estilos de disposición del texto**.

CSS - Texto - Tipos de letra

Estas propiedades son las que afectan principalmente a la **apariencia del texto** (qué tipo de letra se usa, su tamaño, si es negrita, itálica, etc.).

- **font-size**: tamaño de la fuente (px, em, etc.).
- **font-style:** estilo de la fuente (normal, italic, oblique).
- **font-weight**: “ancho” de la fuente (normal, bold, 100-900).
- **font-family**: especifica el tipo de fuente (se pueden setear fuentes ya incluidas en el navegador o importar algunas más “customizadas”).
- **color**: establece el color del texto. Se utilizan valores de *unidad de color*.
- **text-transform**: establece mayúsculas y minúsculas. (uppercase, lowercase, capitalize, none).
- **text-decoration**: línea en el texto (none, underline, overline, line-through).
- **text-shadow**: aplicar sombra a los textos. Se especifican 4 valores (desplazamiento horizontal, desplazamiento vertical, desenfoque, color)

CSS - Texto - Disposición de texto

Propiedades que afectan al espaciado y otras características relativas a la disposición del texto, lo que permite la elección de, por ejemplo, el espacio entre líneas y letras, y el modo como el texto se alinea dentro de la caja contenedora.

- **text-align**: forma en que el texto se alinea dentro del contenedor (left, right, center, justify).
- **line-height:** interlineado. Puede usarse valor con unidad de medida que queramos o, si no se declara la unidad, se usa como multiplicador del font-size.
- **letter-spacing**: espacio entre letras (unidades de medida).
- **word-spacing**: espacio entre palabras.

CSS - Texto - Fuentes Customizadas

Para darle una mejor apariencia a nuestro sitio, podemos incorporar/utilizar fuentes de texto customizadas.

En este sentido, existen muchas librerías donde podemos tomar fuentes gratuitas o bien adquirir su licencia para poder hacer uso de ellas. Google provee una plataforma con innumerables fuentes gratuitas (puede verlas **aquí**).

Una vez seleccionada la fuente que deseemos para nuestro sitio, tendremos diferentes alternativas para incorporarlas dentro de nuestro sitio. A continuación, veremos una de ellas, paso a paso:

Paso 1: En nuestro archivo css general, arriba de todo agregaremos la siguiente línea la cual importa las fuentes directamente desde la plataforma de Google:

```
@import url('https://fonts.googleapis.com/css?family=Roboto');
```

Paso 2: Aplicar la fuente que hemos importado a los elementos que deseemos que se aplique la fuente. Por lo general lo aplicaremos a todos los elementos como en el ejemplo a continuación pero podremos aplicarlo a cualquier selector.

```
* {
    font-family: 'Roboto', sans-serif;
}
```

```
<style>
@import url(
'https://fonts.googleapis.com/css?family=Roboto');
* {
    font-family: 'Roboto', sans-serif;
}
</style>
<h1>Mi página con Custom Fonts</h1>
<p>¡Esta es mi primera página con Custom Fonts!</p>
```

CSS - Íconos

Tal como podemos agregar fuentes customizadas, podemos agregar íconos en nuestra Web, los cuales nos permitirán mejorar el estilo y la interactividad para los usuario. Estos íconos, son tipografías lo que significa que podremos renderizar sin perder calidad del ícono en sí y también podremos aplicarles las mismas propiedades CSS que a los textos (color, alineación, sombras, etc.).

Existen muchas librerías de íconos, una de la más utilizadas es **FontAwesome**, la cual incorporaremos en los siguientes pasos:

Paso 1: Dentro de nuestro **<head>**, importaremos la librería agregando la siguiente línea:

```
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/f
ont-awesome.min.css">
```

Paso 2: Agregar los íconos en nuestro código HTML y dar estilo CSS usando sus clases. Ejemplo:

```
<i class="fa fa-heart"></i>
<i class="fa fa-heartbeat"></i>
```

```
<head>
    <title>Page Title</title>
    <link rel="stylesheet" href="css/index.css">
    <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome.
    <style>
        p i.fa {
            font-size: 48px;
        }

        p {
            color: red;
        }
    </style>
</head>

<body>
    <h1>Mi pàgina con iconos</h1>
    <p>
        <i class="fa fa-heart"></i>
        <i class="fa fa-heartbeat"></i>
    </p>
</body>
```

Ver todos los íconos de la V5 de FontAwesome

CSS - Unidades - Colores

En CSS hay muchas formas de especificar el color. En todas partes en CSS se pueden usar los mismos valores de color, tanto para especificar el color del texto como el color de fondo, o de cualquier otra cosa.

El sistema de colores estándar disponible en los ordenadores modernos es de 24 bits, lo que permite visualizar aproximadamente 16,7 millones de colores distintos a partir de una combinación de diferentes canales de rojo, verde y azul con 256 valores diferentes por canal (256 x 256 x 256 = 16.777.216).

- **Palabras clave**: es la forma más simple de especificar colores (blue, white, black).
- **Valores hexadecimales RGB**: consiste en un símbolo de hashtag/almohadilla (#) seguido de seis cifras hexadecimales (de 0-9 y a-f). Son más complejos, pero más versátiles.
- **Valores RGB y RGBA**: utiliza la función rgb(R, V, A) de css. Los valores van de 0 a 255. RGBA agrega una propiedad extra de opacidad (0-1).
- **Valores HSL y HSLA**: expresa los colores en relación de matiz (0-360), saturación (porcentaje) y luminosidad (porcentaje). HSLA agrega una propiedad extra de opacidad (0-1).

CSS - Listas

Las listas se comportan como cualquier otro texto en su mayor parte, pero hay algunas propiedades CSS específicas de las listas a considerar.

Principalmente los estilos a tener en cuenta pueden agruparse en:

- Espaciado: tamaños de texto, altos de linea, margen y padding.
- Estilo de lista: tipo de lista, posición, imagen de viñeta.

- **list-style-type**: Establece el **tipo de viñetas** para la lista, por ejemplo, viñetas cuadradas o circulares para una lista no ordenada; números, letras, o números romanos para una lista ordenada.
- **list-style-position**: Establece si las viñetas aparecen dentro de los elementos de la lista o fuera de ellos, antes del inicio de cada elemento. (outside, inside)
- **list-style-image**: Te permite usar una imagen personalizada para la viñeta, en lugar de un simple cuadrado o círculo.

CSS - Links

A la hora de dar estilo a los enlaces, es importante comprender cómo utilizar las **pseudo clases** para diseñar los estados de un enlace de manera efectiva y cómo diseñar enlaces para su uso en diversas funciones de interfaz comunes, como menús y pestañas de navegación.

Se considera que un enlace puede tener varios estados.

- **Link**: estado por defecto (:link)
- **Visited**: un enlace que ya ha sido visitado (:visited)
- **Hover**: estado que se activa cuando se le pasa el cursor por encima (:hover)
- **Focus**: estado del enlace en foco. (:focus)
- **Active**: estado de enlace activo, por ejemplo cuando se lo clickea. (:active)

Se puede aplicar un estilo diferente para cada estado distinto del link y brindar una mejor interacción con el usuario

CSS - Links

Algunas propiedades de css de común uso a la hora de darle estilo a los enlaces, en sus diferentes estados son:

- **color**: color del link.
- **cursor**: estilo del cursor.
- **outline**: similar a border, pero que no ocupa tamaño en el box-model.

Se recomienda hacer uso del estilo distintivo entre estados del link, siguiendo alguno de estos lineamientos:

- Subrayar los enlaces, pero no otros elementos. Si esto no es deseado, intentar al menos destacarlos de alguna otra forma.
- Hacer que los links reaccionen de alguna manera cuando se les pasa el cursor por encima, y de una manera algo diferente cuando se activan.

¿Preguntas?

CSS - Pseudo-clases

Todas las pseudo-clases, sin excepción, son una palabra precedida por dos puntos. Las mismas se colocan luego de un selector en particular y nos permiten seleccionar elementos con determinadas características o que se activan por acciones/eventos del usuario.

```
<style>
    p a:hover {
        color: red;
        font-weight: bold;
    }
</style>
<p>
    <a href="/page-1.html">Ir a página 1</a>
</p>
<p>
    <a href="/page-2.html">Ir a página 2</a>
</p>
```

:empty

Con esta pseudo-clase podemos seleccionar los elementos que no tengan hijos ni contenido. El elemento en cuestión debe estar verdaderamente vacío

```
<style>
    p:empty {
        display: none;
    }
</style>
<p></p> <!-- Será seleccionado -->
<p> </p> <!-- No será seleccionado -->
<p>Texto de ejemplo</p> <!-- No será seleccionado -->
```

CSS - Pseudo-clases

:first-child, :last-child y :nth-child

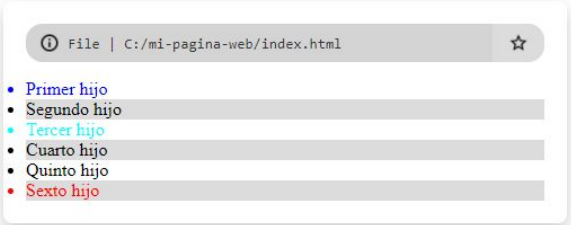
A estas pseudo-clases las utilizamos cuando un elemento tiene dentro N cantidad de otros elementos, es decir, cuando tiene muchos hijos y queremos seleccionar alguno de ellos.

Con **:first-child** seleccionaremos el primer elemento hijo y con **:last-child** el último.

Con **:nth-child(N)** podemos indicar un determinado elemento hijo colocando en lugar de N el número/índice, como así también podemos colocar **fórmulas:**

- **:nth-child(2n)** representa los elementos pares
- **:nth-child(2n+1)** representa los elementos impares
- **:nth-child(5n)** representa los elementos 5, 10, 15…

```
<style>
    ul li:first-child {
        color: blue;
    }
    ul li:last-child {
        color: red;
    }
    ul li:nth-child(3) {
        color: aqua;
    }
    ul li:nth-child(2n) {
        background-color: gainsboro;
    }
</style>
<ul>
    <li>Primer hijo</li>
    <li>Segundo hijo</li>
    <li>Tercer hijo</li>
    <li>Cuarto hijo</li>
    <li>Quinto hijo</li>
    <li>Sexto hijo</li>
</ul>
```



CSS - Pseudo-clases

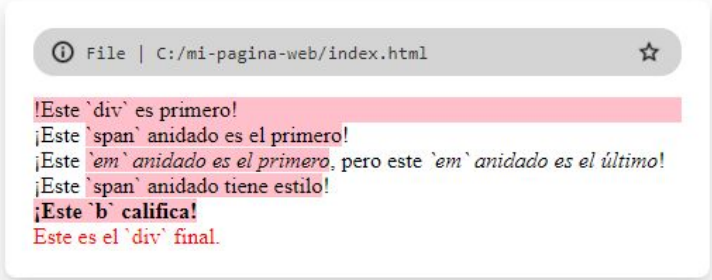
:first-of-type, :last-of-type y :nth-of-type

:first-of-type representa el primer elemento de su tipo entre un grupo de elementos hermanos. Seleccionará el primer elemento dentro de cada elemento que lo contenga descartando a los elementos hermanos. Lo mismo aplica para **:last-of-type** pero con el último elemento.

En el caso de **:nth-of-type(N)**, aplican las mismas reglas que para :nth-child(N) pero para el caso de elementos hermanos.

```
<style>
    article *:first-of-type {
        background-color: pink;
    }

    article div:last-of-type {
        color: red;
    }
</style>
<article>
    <div>!Este `div` es primero!</div>
    <div>¡Este <span>`span` anidado es el primero</span>!</div>
    <div>¡Este <em>`em` anidado es el primero</em>, pero este <em>
`em` anidado es el último</em>!</div>
    <div>¡Este <span>`span` anidado tiene estilo</span>!</div>
    <b>¡Este `b` califica!</b>
    <div>Este es el `div` final.</div>
</article>
```



CSS - Pseudo-clases

Pseudo-clases de hyper-links (enlaces)

:link

:link se refiere a enlaces que no han sido visitados por el usuario aún.

:visited

:visited nos permite estilar enlaces que ya han sido visitados (clickeados) por el usuario.

```
<style>
    a:link {
        color: darkblue;
    }
    a:visited {
        color: gray;
    }
</style>
<a href="/page-1.html">Este enlace ha sido visitado</a><br>
<a href="/page-2.html">Este enlace NO</a>
```

CSS - Pseudo-clases

Pseudo-clases de acción del usuario

:focus

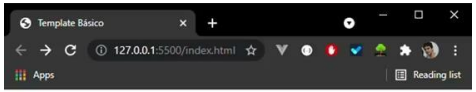
:focus representa un elemento (como una entrada de formulario) que ha recibido el foco. Generalmente se activa cuando el usuario hace clic, toca un elemento o lo selecciona con la tecla "Tab" del teclado.

:hover

:hover se activa cuando el usuario se desplaza sobre un elemento con el cursor (puntero del mouse se posiciona sobre el elemento).

:active

:active hace referencia al evento “click” del usuario sobre un elemento particular.G



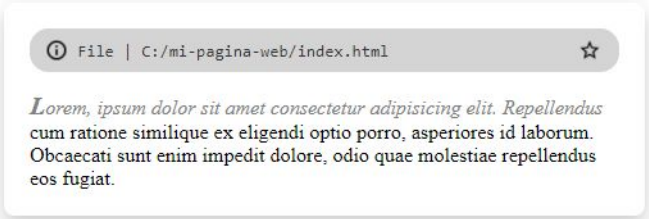

```
<style>
    input#input-txt-example:focus {
        background-color: green;
    }
    input#input-txt-example:hover {
        background-color: red;
    }
    button:active {
        background-color: blue;
        color: white;
    }
</style>
<input id="input-txt-example" type="text">
<button>Enviar</button>
```

Mucho más por explorar: hemos visto las principales pseudo-clases, puede ver el listado completo **aquí**.

CSS - Pseudo-elementos

Se utilizan para diseñar partes específicas de un elemento. A diferencia de las **pseudo-clases**, los **pseudo-elementos** son precedidos por cuatro puntos (**::**).

```
<style>
    p::first-letter {
        font-size: 20px;
        font-weight: bold;
    }
    p::first-line {
        font-style: italic;
        color: gray;
    }
</style>
<p>
Lorem, ipsum dolor sit amet consectetur adipisicing elit. Repellendus cum r
ationе similique ex eligendi optio porro, asperiores id laborum. Obcaecat
i sunt enim impedit dolore, odio quae molestiae repellendus eos fugiat.
</p>
```



::first-letter

Este pseudo-elemento nos permite crear una regla para la primera letra, por ejemplo, la primera letra de un párrafo como se puede ver en el ejemplo anterior.

::first-line

Este pseudo-elemento nos permite crear una regla para la primera línea de un texto, por ejemplo, la primera línea de un párrafo como se puede ver en el ejemplo anterior.

CSS - Pseudo-elementos

::before

::before crea un pseudo-elemento que es el primer hijo del elemento seleccionado. Es usado normalmente para añadir contenido estético a un elemento, usando la propiedad content. *Es en línea (inline) de forma predeterminada.*

::after

::after crea un pseudo-elemento que es el último hijo del elemento seleccionado. Es comúnmente usado para añadir contenido cosmético a un elemento con la propiedad content. *Es en línea (inline) de forma predeterminada*.

```
<style>
    a::before {
        content: "♥";
    }
    a::after {
        content: "→";
    }
</style>
<p><a href="#">Ver más 1</a></p>
<p><a href="#">Ver más 2</a></p>
<p><a href="#">Ver más 3</a></p>
```

Componente toggle: aquí un ejemplo donde se utilizan pseudo-elementos para crear este componente que es muy utilizado.

CSS - Tablas

Cuando se quiere agregar estilo a una tabla, siguiendo diferentes lineamientos, vamos a encontrar que suele haber algunos **patrones** que pueden tomarse para aplicar un mejor diseño. La mayoría utilizan colores o líneas para distinguir diferentes partes de la tabla (por ejemplo los bordes).

Por defecto, el navegador agrega algunos estilos predefinidos para las tablas (user-agent stylesheet). Un ejemplo de esto es la separación de las celdas, utilizando la propiedad **border-spacing**.

Utilizando solo el código CSS mostrado debajo del ejemplo, podemos ver que entre cada celda hay un pequeño espacio de 2px. Ese valor viene determinado dentro de la hoja de estilo del navegador.

Name	ID	Favorite Color
Jim	00001	Blue
Sue	00002	Red
Barb	00003	Green

```css
td, th {
  border: 1px solid #999;
  padding: 0.5rem;
}
```

CSS - Tablas

Por más que pueda modificarse este espacio, lo más normal es eliminarlo. Esto puede lograrse cambiando la propiedad **border-collapse**, aplicando el valor **collapse**.

Name	ID	Favorite Color
Jim	00001	Blue
Sue	00002	Red
Barb	00003	Green

```
table {
  border-collapse: collapse;
}
```

CSS - Tablas

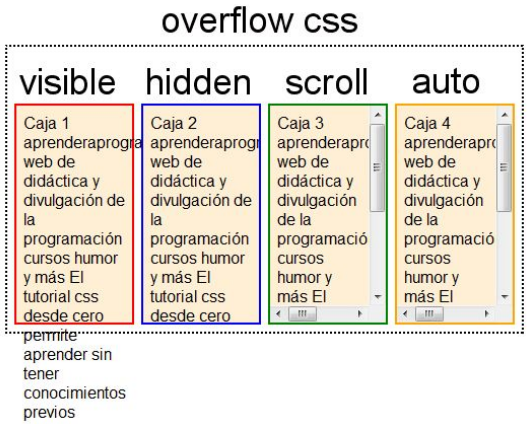
Según la especificación de CSS, es recomendable aplicar estilos de tamano, margin, padding y border a los elementos de celda (td, th) y no a los elementos de fila (tr) dado que el tamaño de las filas depende exclusivamente de sus celdas

Name	ID	Favorite Color
Jim	00001	Blue
Sue	00002	Red
Barb	00003	Green

```css
td, th {
  border: 1px solid #999;
  padding: 0.5rem;
  text-align: left;
}
```

CSS - Overflow

La propiedad **overflow** controla lo que sucede con el contenido que es demasiado grande para caber en un área. Con esto, vamos a especificar si el contenido debe ser recortado en los márgenes del contenedor, o si debe permitirse un scroll dentro del mismo para poder navegar todo el contenido.



Los valores posibles son:

- **visible**: es el valor por defecto. El desbordamiento es visible, por ende no se hace un recorte del contenido y este se renderiza fuera de la caja del elemento.
- **hidden**: el desbordamiento se recorta y el resto del contenido permanece oculto.
- **scroll**: el desbordamiento se recorta y el resto del contenido es visible mediante desplazamiento. Se agregan barras de desplazamiento para manejar la visualización, tanto horizontal como vertical.
- **auto**: similar a scroll, solo que agrega barra de desplazamiento cuando es necesario.

¿Preguntas?

CSS - Posicionamiento - position

La propiedad **position** de CSS especifica cómo un elemento es posicionado en el documento. Las propiedades top, right, bottom, y left determinan la ubicación final de los elementos posicionados.

position: static - valor por defecto. Un elemento con este valor no está posicionado.

Las **propiedades offset** (top, bottom, right y left) se desbloquean con position: **[relative|sticky|absolute|fixed]**

position: relative - se comporta de la misma forma que static a menos que agreguemos las offset properties causando un reajuste para adaptarse a cualquier hueco dejado por el elemento.

position: absolute - el elemento se posiciona en relación a su “ancestro” posicionado (≠ static) más cercano, si no tiene ancestro con esa condición usará el elemento body del documento, y se seguirá moviendo al hacer scroll en la página.

position: fixed - está posicionado con respecto a la ventana del navegador, lo que significa que se mantendrá en el mismo lugar incluso al hacer scroll en la página. No dejará espacio en el lugar de la página donde estaba ubicado normalmente.

position: sticky - se posiciona según la posición de desplazamiento del usuario. Se “pega” en su lugar, luego de alcanzar una posición de desplazamiento determinada.

Para comprender mejor el funcionamiento de **position**, vea **estos ejemplos**.

CSS - Posicionamiento - z-index

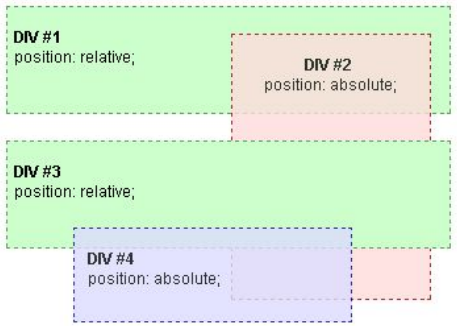
Con la propiedad position y las propiedades offset podremos acomodar los elementos en el plano de dos dimensiones (x/y), inevitablemente nos encontraremos en situaciones en que dos elementos se superponen y es cuando entra en juego la propiedad **z-index**.

Esta propiedad nos permite modificar la forma en que los elementos se superponen entre sí.

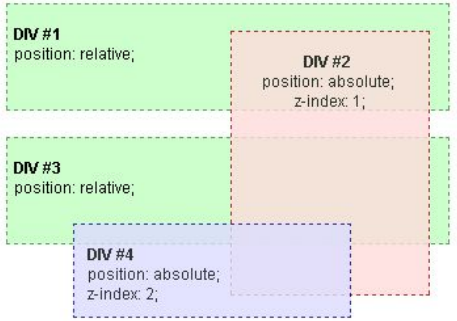
El elemento con valor z-index más alto se propondrá a los que tengan un menor valor.

Como valor para esta propiedad podremos colocarle enteros mayores a 1 teniendo como valor por defecto todos los elementos un **z-index: 0**.

Posicionamiento por defecto:



Aplicando z-index:



CSS - Display

La propiedad **display** es muy (o bien la más) importante para controlar estructuras. Cada elemento tiene un valor de display por defecto, por lo general es “block” (de bloque) o “inline” (en línea).

display: block - un elemento block comienza en una nueva línea y se estira hasta la derecha e izquierda tanto como pueda (ocupa el 100% del width del elemento padre). Ej: , <p>, <h1>, <header>, <footer>.

display: inline - los elementos en línea no ocupan el 100%, si colocamos muchos elementos inline juntos el navegador los posicionará uno al lado del otro. Ej: , <img>, <i>, <a>.

display: none - con este valor el elemento en cuestión se ocultará sin necesidad de eliminarlo.

Otros valores que puede tener display: **list-item**, **table**, **inline-block**, etc.

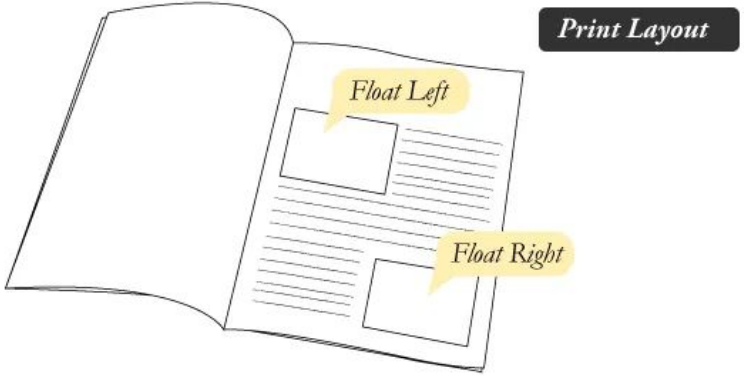
display: flex - El elemento se comporta como un elemento de bloque y establece su contenido (hijos) de acuerdo con el modelo de flexbox (Caja Flexible). Este modelo fue diseñado como un método que pueda ayudar a distribuir el espacio entre los ítems de una interfaz, mejorar las capacidades de alineación y la distribución de los elementos en diferentes tamaños de pantallas. Ejemplos: **ver esta página**

display: grid - El elemento se comporta como un elemento de bloque y establece su contenido (hijos) de acuerdo con el modelo de cuadrícula, es decir, junto con esta propiedad podremos definir columnas y filas en nuestro diseño **grid**. Ejemplos: **ver esta página**

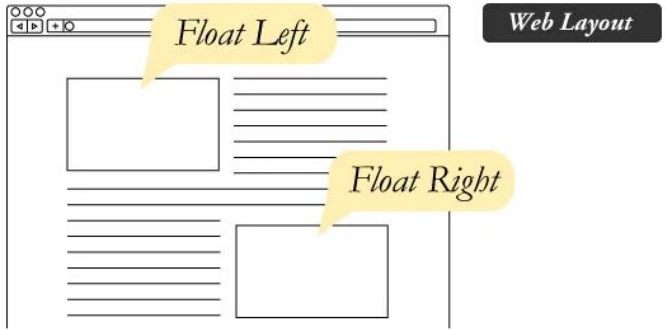
Los modelos **flex y grid** nos permiten obtener un comportamiento parecido al que logramos con las tablas pero con una flexibilidad mucho mayor. Veremos en detalle como funciona en las próximas slides.

CSS - Layouts - Floats

La propiedad **float** fue introducida para permitir a los desarrolladores implementar diseños sencillos que incluyeran una imagen flotando dentro de una columna de texto, con el texto envolviendo la parte izquierda o derecha de la imagen. El tipo de cosa que encuentras habitualmente en el diseño de un periódico.



Puede ser útil también para definir layouts, aunque actualmente una herramienta más adecuada para eso es Flexbox o Grid.



¿Preguntas?

CSS - Flexbox

Flexbox es el nombre corto del módulo CSS de diseño de caja flexible, creado para facilitarnos la disposición de las cosas en una dimensión, ya sea como una fila o como una columna.
Para usar flexbox, debemos aplicar **display: flex** al elemento padre de los elementos que desea diseñar; al aplicar esa declaración todos sus hijos directos se convierten en elementos flexibles.

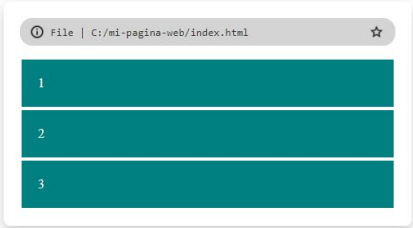
En nuestro ejemplo el **div.container** sería nuestro elemento padre al que le aplicamos la declaración **display: flex**. Los elementos hijos (cada uno de los **div.item**) pasan a ser elementos flexibles.

```
<style>
.container {
    display: flex;
}
.item {
    padding: 20px;
    background-color: teal;
    color: white;
    margin: 2px;
}
</style>
<div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
</div>
```

CSS - Flexbox - flex-direction

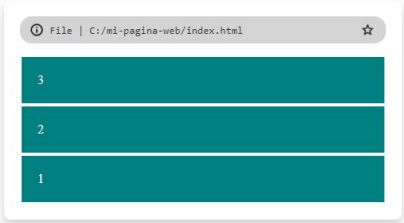
La propiedad **flex-direction** define en qué dirección el contenedor principal acomodará/apilará a los hijos (los flex-items).

flex-direction: **colum** - apila verticalmente los flex-items de arriba hacia abajo.



flex-direction: **row** - **(valor por defecto)** apila los elementos horizontalmente, de izquierda a derecha.

flex-direction: **colum-reverse** - apila verticalmente los flex-items de abajo hacia arriba



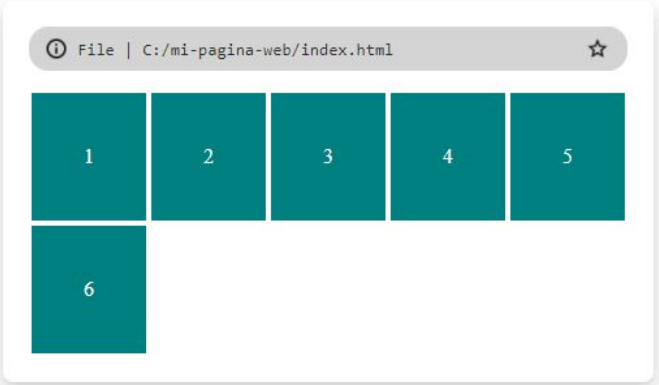
flex-direction: **row-reverse** - apila los elementos horizontalmente, de derecha a izquierda

CSS - Flexbox - flex-wrap

La propiedad **flex-wrap** define si los elementos hijos son obligados a permanecer en una misma línea o si pueden fluir en varias líneas dependiendo del tamaño de la pantalla.

flex-wrap: nowrap - **(valor por defecto)** especifica que los flex-items deben mantenerse en una única línea.

flex-wrap: wrap - permite que los ítems se acomoden en varias líneas, dependiente del tamaño de los propios items como así también del tamaño del contenedor padre y/o de la propia pantalla del dispositivo.



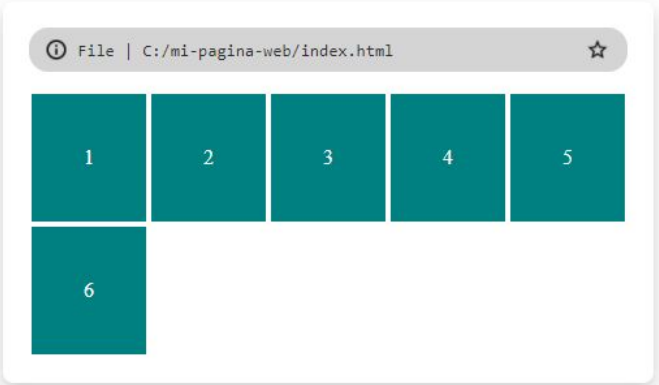
flex-wrap: wrap-reverse - mismo comportamiento que **wrap** con la diferencia que los elementos son ordenador desde el último al primero (para el ejemplo se mostraría 6, 5, 4, 3, 2, 1).

CSS - Flexbox - flex-flow

La propiedad **flex-flow** nos permite establecer las propiedades flex-direction y flex-wrap juntas.

flex-flow: row nowrap - **(valor por defecto)**

flex-flow: row warp



¡Prueba con todas las combinaciones de los valores que hemos visto para flex-direction y flex-wrap!

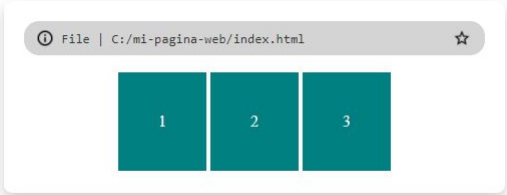
CSS - Flexbox - justify-content

La propiedad **justify-content** nos permite establecer la manera en que los item-flex se distribuirán en el elemento padre.

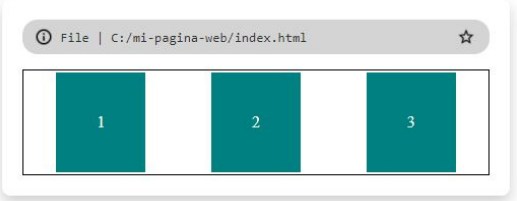
justify-content: flex-start - **(valor por defecto)** ordena los item-flex uno al lado del otro y los alinea a la izquierda.

justify-content: flex-end - alinea los elementos a la derecha.

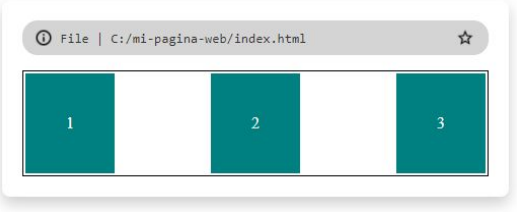
justify-content: center - alinea los elementos en el centro.



justify-content: space-around - distribuye los elementos dejando espacios iguales entre ellos mismos y los bordes del elemento padre.



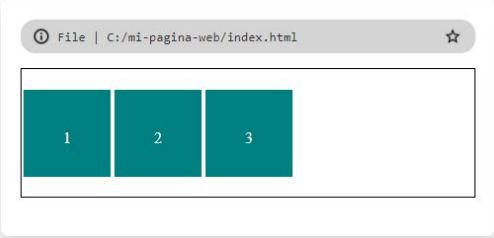
justify-content: space-between - distribuye los elementos sin dejar espacio en los bordes laterales.



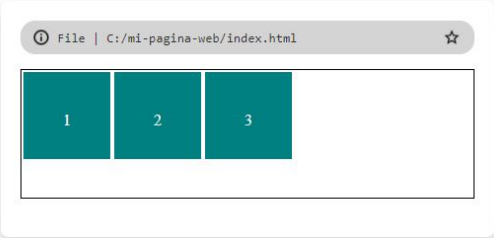
CSS - Flexbox - align-items

La propiedad **align-items** nos permite alinear los ítems verticalmente.

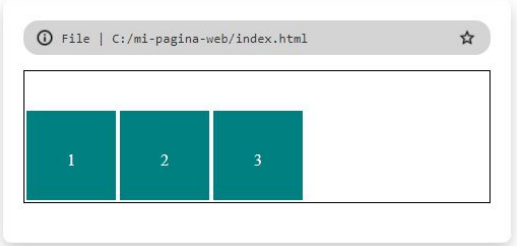
align-items: center - alinea los ítems en el medio del contenedor padre.



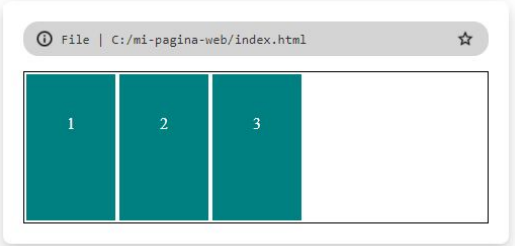
align-items: flex-start - alinea los elementos en la parte superior.



align-items: flex-end - los alinea en la parte inferior.



align-items: stretch - estira los elementos para llenar el alto del contenedor padre.



align-items: baseline - alinea los elementos como sus líneas de base se alinean.

CSS - Flexbox - Centrado perfecto

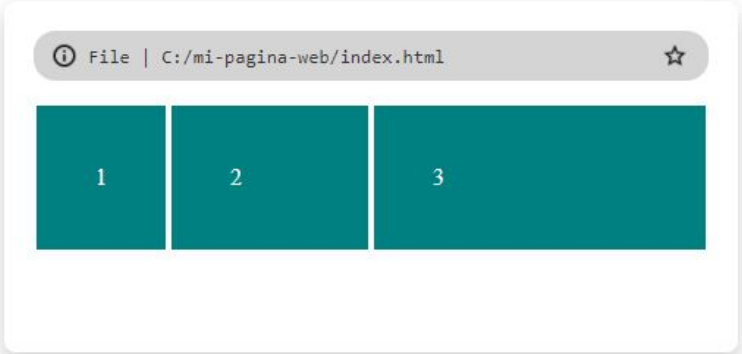
Si queremos lograr un centrado perfecto de un elemento hijo dentro de su elemento padre, con Flexbox es muy sencillo hacerlo. Sólo debemos colocar en el contenedor, las siguientes declaraciones:

```
.container {
    display: flex;
    align-items: center;
    justify-content: center;
}
```

CSS - Flexbox - Propiedades flex-items

flex-grow: nos permite definir cuánto crecerá un elemento en base a los elementos hermanos (otros flex-items). El valor predeterminado es 0

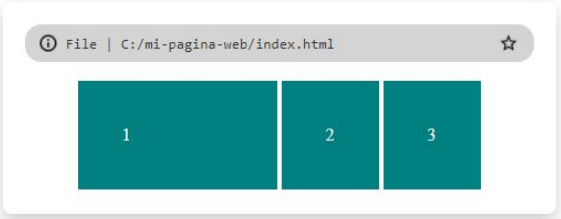
```
<div class="container">
    <div class="item" style="flex-grow: 0;">1</div>
    <div class="item" style="flex-grow: 1;">2</div>
    <div class="item" style="flex-grow: 3;">3</div>
</div>
```



flex-shirnk: nos permite definir cuánto se encogerá un elemento en base a los elementos hermanos (otros flex-items). El valor predeterminado es 0

flex-basis: nos permite especificar la longitud inicial de un flex-item. Los valores de esta propiedad se especifican en px.

```
<div class="container">
    <div class="item" style="flex-basis: 180px;">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
</div>
```

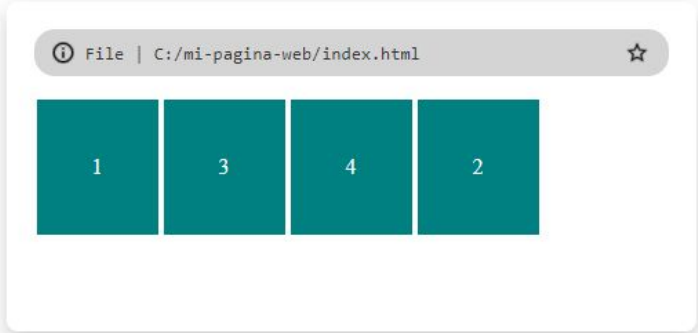


flex: nos permite combinar flex-grow y flex-basis en una misma propiedad

CSS - Flexbox - Propiedades flex-items

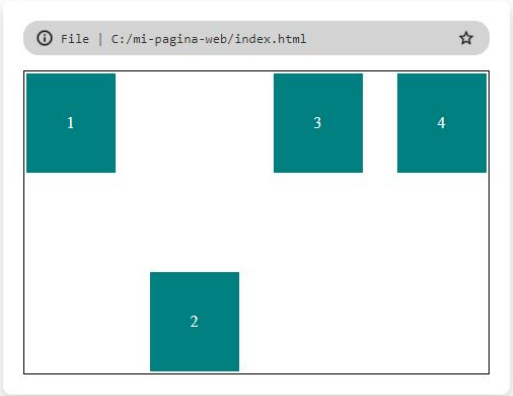
order: esta propiedad nos permite cambiar el orden de un elemento particular. Por ejemplo si queremos que el elemento 2 aparezca último haremos lo siguiente:

```
<div class="container">
    <div class="item">1</div>
    <div class="item" style="order: 1">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
</div>
```



align-self: esta propiedad acepta los mismos valores que para align-items pero se aplica a un elemento puntual.

```
<div class="container">
    <div class="item">1</div>
    <div class="item" style="align-self: flex-end">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
</div>
```



Practica Flexbox con este juego: https://flexboxfroggy.com/#es

¿Preguntas?

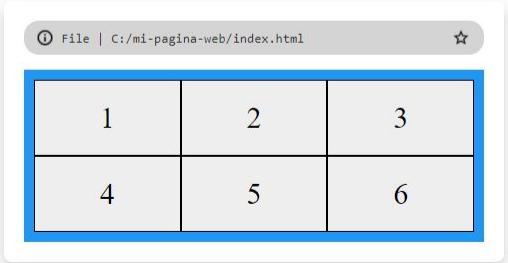
CSS - Grid

CSS grid ofrece un sistema de diseño basado en cuadrículas, con filas y columnas, lo cual facilita el diseño sin tener que usar posicionamiento.
Debemos comenzar definiendo el **display: grid** para convertir el contenedor en una grilla.

> Todos los hijos directos del contenedor de grilla se convierten automáticamente en elementos de grilla (grid-item).

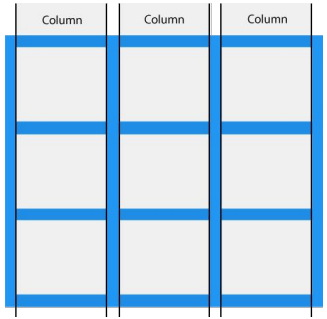
En nuestro ejemplo el **div.container** sería nuestro elemento padre al que le aplicamos la declaración **display: grid** y definimos la cantidad de columnas que tendrá el grid con **grid-template-columns**. Los elementos hijos (cada uno de los **div.item**) pasan a ser elementos grid.

```
<style>
.container {
    display: grid;
    grid-template-columns: auto auto auto;
    background-color: #2196F3;
    padding: 10px;
}
.item {
    background-color: #eee;
    border: 1px solid #000;
    padding: 20px;
    font-size: 30px;
    text-align: center;
}
</style>
<div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
</div>
```

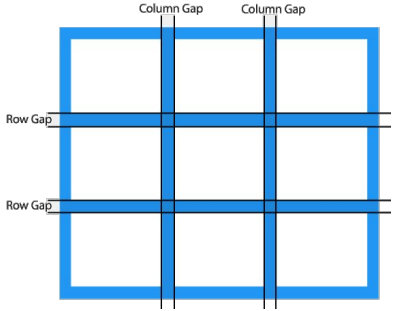


CSS - Grid - Componentes

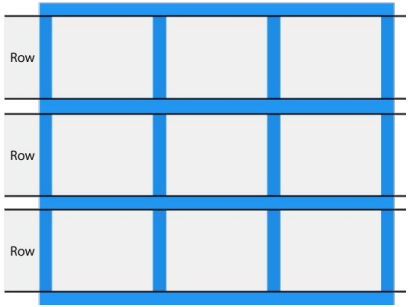
Columnas (grid columns) - las líneas verticales de los elementos de la cuadrícula se denominan columnas.



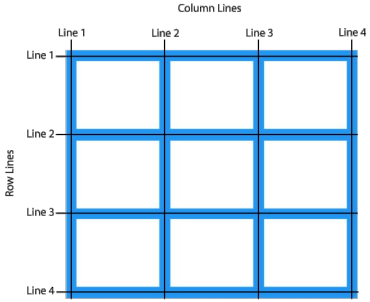
Espacios (grid gaps) - los espacios entre cada columna/fila (o cada elemento) se denomina gap. Podemos ajustarlo con la propiedad **grid-gap**



Filas (grid rows) - las líneas horizontales de los elementos de la cuadrícula se denominan columnas.

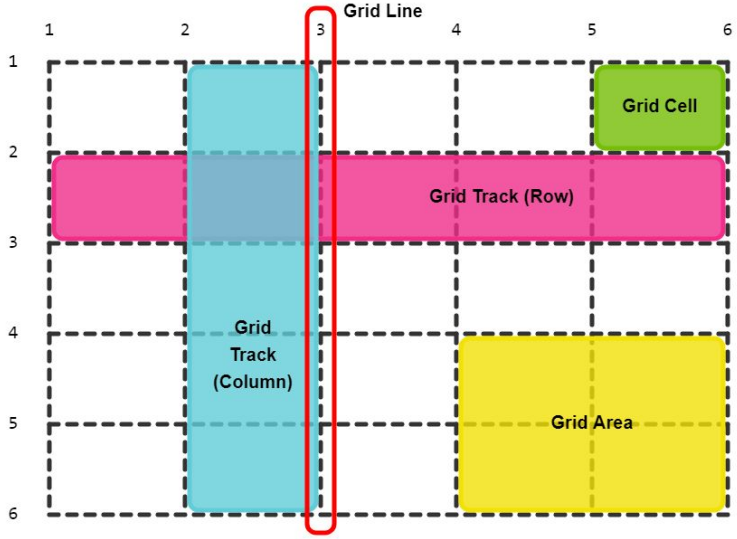


Líneas (grid lines) - las líneas entre cada columna y fila se utilizan como referencia cuando queremos que un elemento ocupe más espacio.



CSS - Grid - Componentes

Grid Area - Al mismo tiempo, podremos definir un área o junto de celdas y ubicarlas en nuestra grilla.

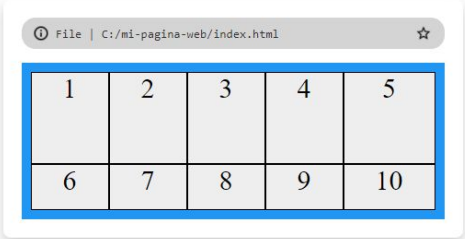


CSS - Grid - Propiedades

grid-template-columns - esta propiedad define el número de columnas en la grilla, y a demás, puede definir el ancho de cada columna.

grid-template-rows - esta propiedad define la altura de cada fila. El valor es una lista separada por espacios.

```
<style>
.container {
    display: grid;
    grid-template-columns: auto auto auto auto auto;
    grid-template-rows: 100px 50px;
}
</style>
<div class="container">
    <div class="item">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
    <div class="item">7</div>
    <div class="item">8</div>
    <div class="item">9</div>
    <div class="item">10</div>
</div>
```



Posibles valores para **grid-template-columns** y **grid-template-row**

- **none**: no hay cuadrícula explícita
- **<absoluta>**: longitud absoluta no negativa (ej: 10px, 2cm).
- **<relativa>**: longitud relativa no negativa (ej: 20%, 3rem).
- **auto**: esta palabra clave nos permite indicar que la columna ocupe el espacio de manera automática, si colocamos a todas las columnas el valor auto las mismas se distribuirán el espacio de manera equitativa.
- **repeat(n, val)**: esta propiedad nos permite establecer valores repetidos. En **n** pondremos el valor de veces a repetir y en **val** el valor a repetir. Por ejemplo “**repeat(6, auto)**” es lo mismo que colocar “**auto auto auto auto auto auto**”.

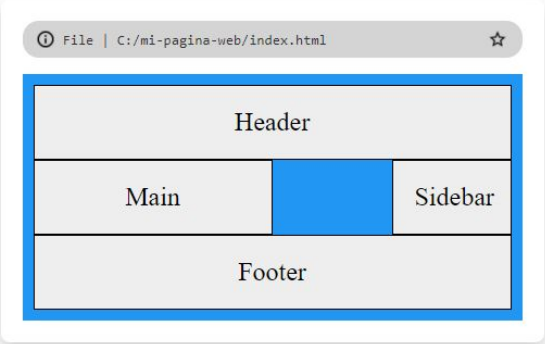
CSS - Grid - Propiedades

grid-template-areas - esta propiedad nos permite organizar los elementos hijos (grid-items) de acuerdo al nombre que le coloquemos al propio elemento en la propiedad **grid-area**. Posible valores:

- **none:** no hay grid áreas definidas.
- **nombre de grid area:** nombre que le hemos definido para el elemento hijo en la propiedad **grid-area**.
- **. (punto):** significa una celda vacía.

grid-area - en esta propiedad colocaremos el nombre que le asignamos al área de nuestra grilla y se coloca a nivel hijo.

```
<style>
.container {
    grid-template-columns: repeat(4, 110px);
    grid-template-areas:
        "header header header header"
        "main main . sidebar"
        "footer footer footer footer";
}
.item-a { grid-area: header; }
.item-b { grid-area: main; }
.item-c { grid-area: sidebar; }
.item-d { grid-area: footer; }
</style>
<div class="container">
    <div class="item item-a">Header</div>
    <div class="item item-b">Main</div>
    <div class="item item-c">Sidebar</div>
    <div class="item item-d">Footer</div>
</div>
```



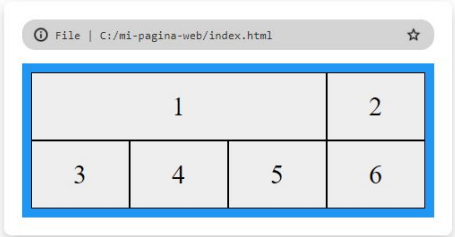
CSS - Grid - Propiedades grid-items

grid-column-start: línea donde la columna comienza.

grid-column-end: línea donde la columna termina.

grid-column: propiedad abreviada para ambas anteriores. Se colocan los valores start y end de la siguiente manera: **[start]** / **[end]**.

```
<style>
.item-a {
    grid-column: 1 / 4;
}
</style>
<div class="container">
    <div class="item item-a">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
</div>
```

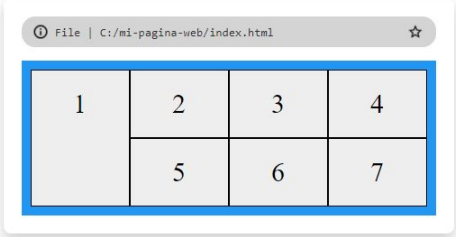


grid-row-start: línea donde la columna comienza.

grid-row-end: línea donde la columna termina.

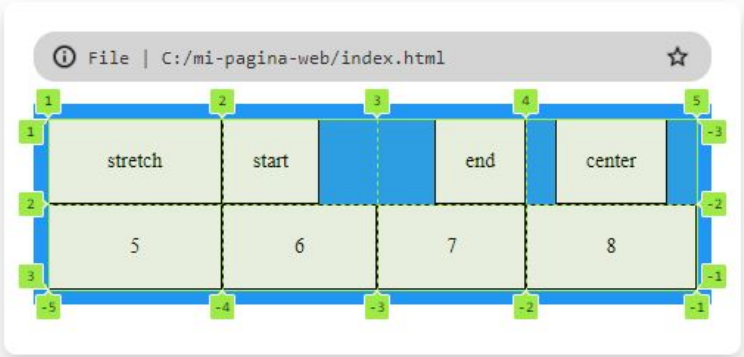
grid-row: propiedad abreviada para ambas anteriores. Se colocan los valores start y end de la siguiente manera: **[start]** / **[end]**.

```
<style>
.item-a {
    grid-row: 1 / 3;
}
</style>
<div class="container">
    <div class="item item-a">1</div>
    <div class="item">2</div>
    <div class="item">3</div>
    <div class="item">4</div>
    <div class="item">5</div>
    <div class="item">6</div>
    <div class="item">7</div>
</div>
```

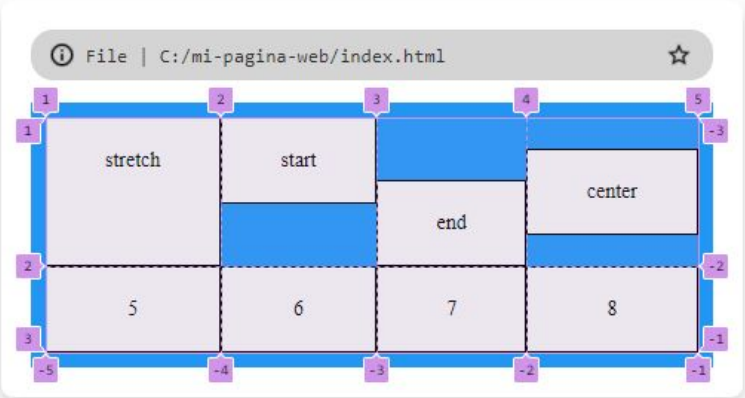


CSS - Grid - Propiedades grid-items

justify-self: alínea un grid-item dentro de una celda a lo largo de la fila. Posibles valores: **stretch** (por defecto), **start**, **end**, **center**.

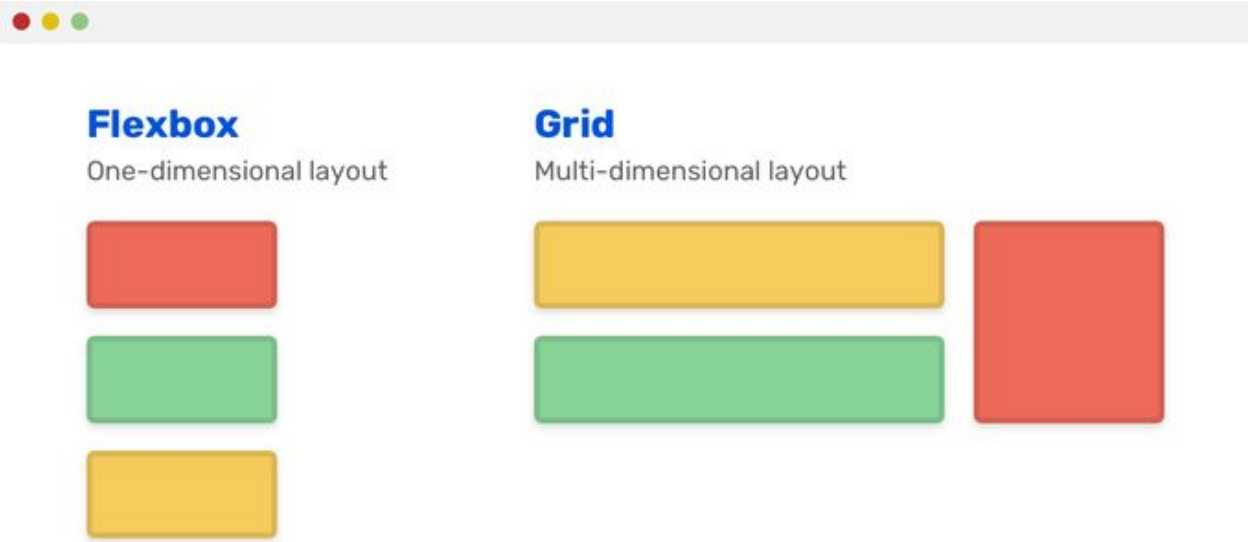


align-self: alínea un grid-item dentro de una celda a lo largo de la columna. Posibles valores: **stretch** (por defecto), **start**, **end**, **center**.



Practica Grid con este juego: https://cssgridgarden.com/#es

CSS - Flexbox vs Grid



Normalmente utilizaremos **Grid** para construir **layouts** y **Flexbox** para **componentes**.

¿Preguntas?

CSS - Media queries

Las **media queries** (en español "consultas de medios") son útiles cuando deseas modificar tu página web o aplicación en función del tipo de dispositivo (como una impresora o una pantalla) o de características y parámetros específicos (como la resolución de la pantalla o el ancho del viewport del navegador).
Las **media queries** se pueden usar para verificar muchas cosas, como:

- ancho y alto de la ventana gráfica
- ancho y alto del dispositivo
- orientación (¿la tableta / teléfono está en modo horizontal o vertical?)
- resolución

Una **media queries** consta de un tipo de medio y puede contener una o más expresiones, que se resuelven en verdadero o falso. Cuando se resuelve en **verdadero**, el bloque de código CSS que coloquemos dentro se aplicará mientras que cuando se resuelve **falso** no.

```
@media not|only mediatype and (expressions) {
  CSS-Code;
}
```

También puede tener diferentes hojas de estilo para diferentes medios:

```
<link rel="stylesheet" media="mediatype and|not|only (expressions)" href="print.css">
```

Con esta herramienta podremos hacer nuestro sitio **responsivo** (responsive), es decir, que se visualice correctamente en cualquier dispositivo.

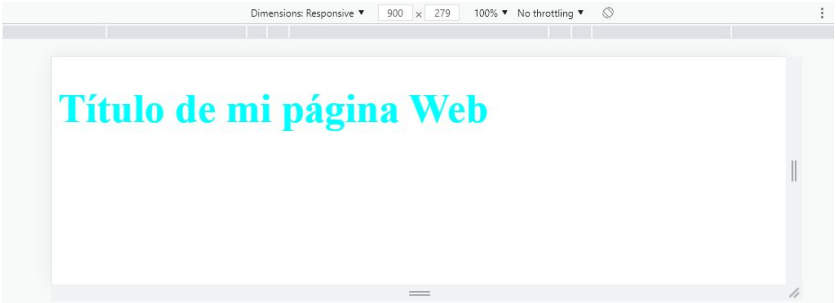
CSS - Media queries - Ejemplo

A continuación, podremos ver un ejemplo sencillo e ilustrativo del uso de media-queries en un h1 pero tengamos en cuenta que podremos aplicar cualquier estilo css dentro de las media-queries.

```
<style>
h1 {
    font-size: 50px;
    color: aqua
}

@media screen and (max-width: 480px) {
    h1 {
        font-size: 25px;
        color: brown;
    }
}
</style>
<h1>Título de mi página Web</h1>
```

Ancho de pantalla (width): 900px



Ancho de pantalla (width): 400px

CSS - Variables

Las variables en CSS nos permiten almacenar valores de propiedades y utilizarlos cuantas veces lo deseemos.
Las mismas se debe declarar dentro de un selector, generalmente utilizaremos **:root** para que la variable sea global.
El nombre de una variable CSS debe comenzar con dos guiones (“--”) y se distinguen mayúsculas de minúsculas.
Una vez declarada la variable, para utilizarla o instanciar esa variable en cualquier parte de nuestro código CSS, debemos hacerlo de usando la palabra reservada **var()**.

```
<style>
:root {
    --main-color: #5f9ea0;
}

h1 {
    color: var(--main-color);
}

hr {
    border-color: var(--main-color);
}

</style>
<h1>Título de mi página Web</h1>
<hr />
```