CSS3

Introducción

Hojas de Estilo en Cascada (del inglés Cascading Style Sheets) o CSS es el lenguaje de estilos utilizado para describir la presentación de documentos HTML o XML.

Esquetelo básico

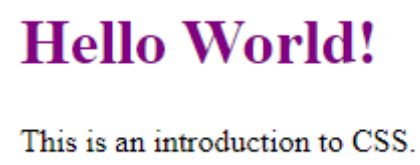
Todo lo que se hace en CSS sigue este patrón básico:

```
selector {
    property: value;
}
```

Ejemplo:

Hacer que todos los elementos <h1> sean de color violeta:

```
h1 {
    color: purple;
}
```



¿Cómo incluir los estilos?

Hay tres formas de estilizar un documento HTML:

Los estilos pueden incluirse en la misma línea del elemento HTML que queremos estilizar (esta es la peor opción por razones obvias).

```
<body>
    <h1 style="color:purple">Hello World!</h1>
    <p>This is an introduction to CSS.</p>
</body>
```

También se pueden incluir los estilos dentro de la etiqueta <style></style> dentro del elemento <head> del documento HTML. (Mejor que la anterior, pero dificulta compartir estilos con otros documentos).

```
    <style>
        h1 {
            color: purple;
        }
    </style>
```

La tercera opción es crear una hoja de estilos externa (opción recomendada). Se crea un archivo de extensión .css y esta se vincula al documento HTML. La manera de vincular ambos documentos es mediante el elemento **<link>**.

El elemento HTML <link>

El elemento HTML <link> especifica la relación entre el documento actual y un recurso externo. Se usa de la siguiente manera:

```
<head>
    <title>CSS Intro</title>
    <link rel="stylesheet" href="prueba.css">
</head>
```

rel

Este atributo indica la relación del documento enlazado con el actual. El atributo debe ser una lista de tipos de enlaces separados por espacio. El uso más común para este atributo es especificar el enlace a una hoja de estilos externa: el atributo ***rel*** se establece con valor ***stylesheet***, y el atributo **href** se establece con la URL/URI de la hoja de estilos externa para dar formato a la página.

Algunas propiedades básicas (color)

Color

La propiedad de CSS color selecciona el valor de color de primer plano del contenido de elemento de texto y decoraciones de texto. También establece el valor <currentcolor> que se puede usar como un valor indirecto en otras propiedades, y es el valor predeterminado para otras propiedades de color, como border-color.

Con respecto al color del primer plano deberás tener en cuenta que:

- Si añades color al primer plano de una imagen, ésta seguirá viéndose, pero el color se aplicará al borde de la imagen.
- La propiedad **border-color** ignora la propiedad color.
- Para configurar el color de todo un documento debemos escribir una regla con esta propiedad color para el selector body.
- La aplicación de color a los elementos de los formularios no funciona bien en todos los navegadores.

Ejemplo

A continuación, todas las formas de hacer que el atributo color de todos los elementos párrafo sean rojo:

```
p {
    color: red;
}

p {
    color: #f00;
}

p {
```

```
    color: #ff0000;
}

p {
    color: rgb(255, 0, 0);
}

p {
    color: rgb(100%, 0%, 0%);
}

p {
    color: hsl(0, 100%, 50%);
}

/* 50% translúcido */
p {
    color: rgba(255, 0, 0, 0.5);
}

p {
    color: hsla(0, 100%, 50%, 0.5);
}
```

Sistemas de colores

Colores por nombre

Los navegadores modernos soportan unos 140 nombres de colores, pueden verse todos en la siguiente lista:

https://htmlcolorcodes.com/color-names/

RGB

RGB es un modelo de color basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores de luz primarios.

En CSS se usa de la siguiente manera:

Tenemos tres canales: R (red), G (green) y B (blue). Cada canal acepta valores en el rango: 0-255, donde 0 sería nada de color y 255 su valor máximo.

Ejemplo de sintaxis

Color rojo: **rgb(255, 0, 0)**

También se pueden usar porcentajes de color donde 0% sería 0 y 100% 255.

RGBA

Igual que RGB pero con un canal adicional (canal ***alpha***) que nos da información sobre la opacidad en el rango [0,1], donde 0 es totalmente traslúcido y 1 es totalmente opaco.

Ejemplo de sintaxis

Color rojo: **rgba(255, 0, 0,1)**

Hexadecimal

Seguimos teniendo 3 canales de color, también en el rango 0-255, pero en este caso los representamos con notación hexadecimal. El sistema hexadecimal (abreviado hex.) es el sistema de numeración posicional que tiene como base el 16. En principio, dado que el sistema usual de numeración es de base decimal y, por ello, solo se dispone de diez dígitos, se adoptó la convención de usar las seis primeras letras del alfabeto latino para suplir los dígitos que faltan. El conjunto de símbolos es el siguiente: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}.

En CSS seguiremos el mismo orden para los canales que en RGB, donde esta vez el rango es 0-F. Emplearemos un valor que se compone de 6 dígitos hexadecimales en grupos de tres. Los dos primeros representan el canal rojo, los siguientes dos el verde y los dos últimos el azul. Los símbolos se escriben todos juntos después del símbolo almohadilla: #

Ejemplo de sintaxis

Color rojo: #ff0000

También se puede escribir:

Color rojo: #f00

Esto solo es válido cuando las duplas de cada canal son iguales.

HSL

El modelo HSL (Hue, Saturation, Lightness; o en español: matiz, saturación, luminosidad), también llamado HSI (Hue, Saturation, Intensity), define un modelo de color en términos de sus componentes constituyentes.

En CSS se emplean porcentajes (0-100%, canales S y L) o grados (solo para le canal H). El canal H (matiz) representaría al color en sí mismo, cada valor es un grado (0-360) en el círculo cromático. 0 sería rojo, 120 verde y 240 sería azul. En el Canal S (saturación), 0% sería en escala de grises y 100% sería el color al total de saturación. Por último, el canal L (luminosidad), donde 0% sería negro y 100% sería blanco.

Ejemplo de sintaxis:

Color rojo: **hsl(0, 100%, 50%)**

HSLA

Al igual que con RGBA se añade un canal extra alpha, se usa de la misma forma que en RBGA.

Enlace útil: https://htmlcolorcodes.com/color-picker/

Opacidades

La opacidad es una característica de los elementos que nos permite mostrar o no otros elementos que tengan por debajo. Para conseguir efectos de transparencia enalgunos elementos tienes las siguientes propiedades: **opacity, moz-opacity y filter.**

opacity

Esta propiedad, que es compatible con todos los navegadores que soporten CSS3, permite asignar valores comprendidos entre 0 (invisible o totalmente transparente) y 1 (totalmente opaco).

-moz-opacity

Esta propiedad, permite asignar los mismos valores que la propiedad anterior. La diferencia está en que sólo es compatible con versiones anteriores del Firefox 0.9.

filter

Esta propiedad, de IE (5.5 y siguientes), tiene varios efectos: degradaciones, desenfocados, sombras, etcétera. Para lograr la transparencia hay que aplicar el filtro alpha, con valores entre 0 y 100.

```
#saludo {
    background: white;
    width: 200px;
    height: 200px;
    filter: alpha(opacity=50);
    -moz-opacity: 0.5;
    opacity: 0.5;
}
```

background-color

Es una propiedad de CSS que define el color de fondo de un elemento, puede ser el valor de un color o la palabra clave transparent.

Las propiedades relativas al fondo no se heredan, pero, como el valor predeterminado de esta propiedad es transparent, salvo que se especifique un color concreto, el color de fondo del elemento padre aparecerá a través de sus elementos hijos.

Ejemplo

```
h1 {
    background-color: red;
}
```

This is an introduction to CSS.

Algunas propiedades básicas (texto)

text-align

La propiedad text-align de CSS establece la alineación horizontal del contenido a nivel de línea dentro de un elemento de bloque o caja de celda-tabla. Esto significa que funciona como vertical-align pero en dirección horizontal.

Sintaxis y valores

```
/* Valores clave */
text-align: start;
/*Lo mismo que left si la dirección es de izquierda a derecha y right si
la dirección es de derecha a izquierda.*/
text-align: end;
```

```
/*Lo mismo que right si la dirección es de izquierda a derecha e left si
la dirección es de derecha a izquierda.*/
text-align: left;
/*El contenido en línea se alinea con el borde izquierdo de la línea de la
caja.*/
text-align: right;
text-align: center;
text-align: justify;
text-align: justify-all;
/*Igual que justify, pero también obliga a justificar la última línea.*/
text-align: match-parent;

/* Alineación basada en caracteres en una columna de la tabla */
text-align: ".";
text-align: "." center;

/* Valores de alineación de elementos de bloque (Sintaxis no estándar) */
text-align: -moz-center;
text-align: -webkit-center;

/* Valores globales */
text-align: inherit;
text-align: initial;
text-align: revert;
text-align: revert-layer;
text-align: unset;
```

font-weight

La propiedad font-weight de CSS especifica el peso o grueso de la letra. Algunos tipos de letra sólo están disponibles en normal y bold.

Sintaxis y valores

```
font-weight: normal;
font-weight: bold;

/* Relativo al padre */
font-weight: lighter;
font-weight: bolder;

font-weight: 100;
font-weight: 200;
font-weight: 300;
font-weight: 400;
font-weight: 500;
font-weight: 600;
font-weight: 700;
font-weight: 800;
font-weight: 900;

/* Valores globales */
font-weight: inherit;
```

```
font-weight: initial;
font-weight: unset;
```

text-decoration

La propiedad CSS text-decoration se usa para establecer el formato de texto subrayado (**underline**) y suprarrayado (**overline**), tachado (**line-through**) o parpadeo (**blink**). El subrayado y el suprarrayado son posicionados bajo el texto, mientras la línea a través de éste se posiciona por encima.

Sintaxis, valores y ejemplos

```
/* Valores clave */
text-decoration: none; /* Sin decoración */
text-decoration: underline red; /* Subrayado rojo */
text-decoration: underline wavy red; /* Subrayado rojo ondulado */

/* Valores globales */
text-decoration: inherit;
text-decoration: initial;
text-decoration: unset;
```

Propiedades específicas:

text-decoration-color

text-decoration-style

```
/* Valores clave */
text-decoration-style: solid;
text-decoration-style: double;
text-decoration-style: dotted;
text-decoration-style: dashed;
text-decoration-style: wavy;

/* Valores globales */
text-decoration-style: inherit;
text-decoration-style: initial;
text-decoration-style: unset;
```

text-decoration-line

```
/* Palabra clave única */
text-decoration-line: none;
text-decoration-line: underline;
text-decoration-line: overline;
text-decoration-line: line-through;
text-decoration-line: blink;

/* Varias palabras clave */
text-decoration-line: underline overline; /* Dos líneas de decoración */
text-decoration-line: overline underline line-through; /* Múltiples líneas
de decoración */
```

```
/* Valores globales */
text-decoration-line: inherit;
text-decoration-line: initial;
text-decoration-line: revert;
text-decoration-line: revert-layer;
text-decoration-line: unset;
```

text-decoration-thickness.

```
/* Single keyword */
text-decoration-thickness: auto;
text-decoration-thickness: from-font;

/* length */
text-decoration-thickness: 0.1em;
text-decoration-thickness: 3px;

/* percentage */
text-decoration-thickness: 10%;

/* Global values */
text-decoration-thickness: inherit;
text-decoration-thickness: initial;
text-decoration-thickness: revert;
text-decoration-thickness: revert-layer;
text-decoration-thickness: unset;
```

text-transform

Controla la apariencia de las letras en un elemento. Los valores que puede tener son: **none** (texto normal, con mayúsculas y minúsculas), **capitalize** (cada palabra comienza con mayúsculas), **uppercase** (todo el texto aparece en mayúsculas) y **lowercase** (todo el texto aparece en minúsculas).

```
    text-transform: uppercase;
```

line-height

La propiedad CSS line-height establece la altura de una casilla remarcada por líneas. Comúnmente se usa para establecer la distancia entre líneas de texto. A nivel de elementos de bloque, define la altura mínima de las casillas encuadradas por líneas dentro del elemento. En elementos en línea no reemplazables, especifica la altura que se usa para calcular la altura de la casilla encuadrada por líneas.

Sintaxis, valores y ejemplos

```
/* Keyword value */
line-height: normal;

/* Unitless values: usa esta cifra multiplicada por el tamaño de fuente
del elemento */
line-height: 3.5;
```

```
/* <longitud> valores */
line-height: 3em;

/* <porcentaje> valores */
line-height: 34%;

/* Valores absolutos */
line-height: inherit;
line-height: initial;
line-height: unset;
```

text-indent

Sangra la primera línea de texto de un párrafo.

letter-spacing

Configura sobre el espacio que hay entre los caracteres. Este valor puede aumentar o disminuir ya que, al igual que text-indent y otras propiedades, admite valores positivos y negativos.

white-space

Permite establecer cómo se gestionan los espacios en blanco en un elemento. Los valores que puede tener son: **normal** (los espacios en blanco adicionales son ignorados por el navegador), **pre** (los espacios en blanco adicionales son utilizados como cuando se emplea la etiqueta pre en HTML), **nowrap** (no se produce el ajuste de línea automático por lo que el texto permanecerá en la misma línea hasta que encuentre una etiqueta
).

font

La propiedad abreviada CSS de fuente establece todas las diferentes propiedades de la fuente de un elemento. Alternativamente, establece la fuente de un elemento en una fuente del sistema.

Esta propiedad es una abreviatura de las siguientes propiedades CSS:

- font-family
- font-size
- font-stretch
- font-style
- font-variant
- font-weight
- line-height

Sintaxis

```
/* font-size font-family */
font: 1.2em "Fira Sans", sans-serif;

/* font-size/line height font-family */
font: 1.2em/2 "Fira Sans", sans-serif;

/* font-style font-weight font-size font-family */
font: italic bold 1.2em "Fira Sans", sans-serif;

/* font-stretch font-variant font-size font-family */
```

```
font: ultra-condensed small-caps 1.2em "Fira Sans", sans-serif;

/* system font */
font: caption;
```

Ejemplo

```
h1 {
    text-transform: uppercase;
    font-family:monospace;
    text-align:center;
    font-weight:100;
    font-size:40px;
    letter-spacing:20px;
    text-decoration: plum underline wavy;
}
```

Font-size

La propiedad font-size especifica la dimensión de la letra. Este tamaño puede, a su vez, alterar el aspecto de alguna otra cosa, ya que se usa para calcular la longitud de las unidades em y ex.

Sintaxis y ejemplos

```
/* <absolute-size> values */
font-size: xx-small;
font-size: x-small;
font-size: small;
font-size: medium;
font-size: large;
font-size: x-large;
font-size: xx-large;
font-size: xxx-large;

/* <relative-size> values */
font-size: smaller;
font-size: larger;

/* <length> values */
font-size: 12px;
font-size: 0.8em;

/* <percentage> values */
font-size: 80%;

/* math value */
font-size: math;

/* Global values */
font-size: inherit;
font-size: initial;
```

```
font-size: revert;
font-size: revert-layer;
font-size: unset;
```

Font-family

La propiedad font-family define una lista de fuentes o familias de fuentes, con un orden de prioridad, para utilizar en un elemento seleccionado. A diferencia de la mayoría de las propiedades CSS, los valores se separan con comas (",") para indicar que son valores alternativos.

```
/* A font family name and a generic family name */
font-family: "Gill Sans Extrabold", sans-serif;
font-family: "Goudy Bookletter 1911", sans-serif;

/* A generic family name only */
font-family: serif;
font-family: sans-serif;
font-family: monospace;
font-family: cursive;
font-family: fantasy;
font-family: system-ui;
font-family: ui-serif;
font-family: ui-sans-serif;
font-family: ui-monospace;
font-family: ui-rounded;
font-family: emoji;
font-family: math;
font-family: fangsong;

/* Global values */
font-family: inherit;
font-family: initial;
font-family: revert;
font-family: revert-layer;
font-family: unset;
```

Siempre se debe incluir al menos un nombre de familia genérico en una lista de familias de fuentes, ya que no hay garantía de que una fuente determinada esté disponible. Esto permite que el navegador seleccione una fuente alternativa aceptable cuando sea necesario.

La propiedad font-family especifica una lista de fuentes, desde la prioridad más alta hasta la más baja. La selección de fuentes no se detiene en la primera fuente de la lista que se encuentra en el sistema del usuario. Más bien, la selección de fuentes se realiza de a un carácter a la vez, de modo que, si una fuente disponible no tiene un glifo para un carácter necesario, se prueba con las últimas fuentes. Cuando una fuente solo está disponible en algunos estilos, variantes o tamaños, esas propiedades también pueden influir en la familia de fuentes que se elige.

Enlace de interés: https://www.cssfontstack.com/

Font-style

Nos permite configurar el "estilo" de la fuente. Hay tres valores posibles: normal que no configura ningún estilo enparticular sino que toma el definido por defecto en el navegador, italic: que equivale al elemento del HTML clásico(<i>) que coloca el testo en cursiva y oblique que funciona, aparentemente, como "italic".

Font-variant

Permite dos posibilidades: **normal y small-caps**. Con normal, el texto no cambia de apariencia y con small-caps eltexto pasa a mostrarse en mayúsculas de un tamaño inferior.

Selectores CSS

Selector universal

Selecciona todos los elementos.

```
* {
    color: black;
}
```

Selector elemento

Selecciona todos los elementos de determinado tipo.

```
img {
    width: 200px;
    height: 100px;
}
```

Selectores múltiples

Se pueden combinar distintos selectores separándolos mediante una coma.

```
h1, h2 {
    color: black;
}
```

Selector ID

Selecciona los elementos con cierto id. El id se referencia con el símbolo # delante del nombre (#nombre_id)

```
#logout {
    color: orange;
    height: 200px;
}
```

Enlace de interés: https://coolors.co/palettes/trending

Se recomienda que el valor del id sea un nombre que caracterice o clarifique, de forma breve y esquemática al elemento y que, además, sea fácilmente reconocible por el programador. Se utilizan con frecuencia para identificar las secciones principales de las páginas**: contenido, cabecera, pie**, etcétera.

Selector clase

El **atributo global de clase** es una lista separada por espacios de las clases del elemento que distinguen entre mayúsculas y minúsculas. Las clases permiten que CSS y JavaScript seleccionen y accedan a elementos específicos a través de selectores de clase o funciones como el método DOM document.getElementsByClassName.

En el HTML:

```
<body>
    <h1 class="titles">Hello World!</h1>
    <p>This is an introduction to CSS.</p>
</body>
```

En el CSS:

```
.titles {
    color: orange;
}

/*Seleccionamos todas las clases titles*/

h1.titles {
    color: orange;
}

/*Seleccionamos todas las clases titles que estén dentro del selector h1*/
```

Este selector es útil para estilizar elementos que se repiten, como parte de un grupo conceptual. Así, los elementos de una clase pueden modificarse con una única regla de estilo. También se puede hacer que un elemento pertenezca a más de una clase separando sus nombres de clase con espacios.

Selector de descendientes

Se escribe separado por un espacio en primer lugar el elemento padre seguido de los elementos hijos. Por ejemplo:

```
span a {
    background-color:#ffffff;
}
```

Selecciona todos los elementos <a> que estén anidados dentro de elementos

Selector descendiente directo o hijo

```
/*Como el anterior pero solo para el descendiente DIRECTO*/
footer > a {
    color:red;
}
```

Selecciona los elementos <a> que sean descendientes directos de elementos <footer>

En el HTML

```
    <footer>
        <nav>
            <ul>
                <li>
                    <a href=" #home">Home</a>
                </li>
                <li>
                    <a href="#about">About</a>
                </li>
                <li>
                    <a href="#contact">Contact</a>
                </li>
                <li>
                    <a href="http:\\www.google.es">Google</a>
                </li>
            </ul>
        </nav>
        <a href="""">Content is available under these licenses.</a>
    </footer>
```

Aquí solo estaríamos estilizando el último <a>, pues es el único descendiente directo del elemento <footer>

Selector adyacente

Se usa utilizando el símbolo "+" de la siguiente manera: elemento + el elemento adyacente (inferior en un sentido no jerárquico, solo siguiendo el sentido de la "cascada"), que es el que queremos estilizar. (No implica jerarquía/anidamiento, ya que input y button son hermanos)

```
/*Selecciona solo los elementos "botón" que están inmediatamente precedidos
por un "input"*/
input + button {
    background-color: black;
    color: white;
}
```

En el HTML

```
        <input type="password" placeholder="Password" id="passwd">
        <button>Log In</button>
```

Estaríamos estilizando ese botón. Nótese que ambos elementos van seguidos (misma jerarquía), son adyacentes, pero no están anidados (jerarquía padre/hijo).

Selector general hermano (~)

Es similar al selector adyacente " + " solo que, mientras el segundo solo se aplica al hermano inmediatamente posterior (en el sentido de la cascada), el segundo selecciona a cualquier hermano.

```
<nav class="navbar">
        <!--Menú hamburguesa-->
        <input type="checkbox" class="menu-toggle" />
        <svg xmlns="http://www.w3.org/2000/svg" class="burger-icon"
viewBox="0 0 16 16">
            <path fill-rule="evenodd"
                d="M2.5 12a.5.5 0 0 1 .5-.5h10a.5.5 0 0 1 0 1H3a.5.5 0 0
1-.5-.5m0-4a.5.5 0 0 1 .5-.5h10a.5.5 0 0 1 0 1H3a.5.5 0 0 1-.5-.5m0-
4a.5.5 0 0 1 .5-.5h10a.5.5 0 0 1 0 1H3a.5.5 0 0 1-.5-.5" />
        </svg>
        <!--Fin menú hamburguesa-->
        <ul class="navbar-list">
         .
         .
         .
```

```
/*Cuando se haga click en el botón de check (icono hamburguesa) se
muestra .navbar-list*/
.menu-toggle:checked~.navbar-list {
    display: block;
}
```

Selector atributo

Selecciona todos los elementos de cierto tipo, para los cuales determinado atributo cumple con el valor especificado entre comillas.

Sintaxis:

```
elemento[atributo="valor"] {
    /*Propiedades*/
}
```

Por ejemplo:

```
/*Selecciona todos los elementos input con el atributo "password"*/
input[type="password"] {
    color:red;
}
```

Otras sintaxis que se pueden usar:

```
/* <a> elements with a title attribute */
a[title] {
    color: purple;
}

/* <a> elements with an href matching "https://example.org" */
a[href="https://example.org"] {
```

```
    color: green;
}

/* <a> elements with an href containing "example" */
a[href*="example"] {
    font-size: 2em;
}

/* <a> elements with an href ending ".org", case-insensitive */
a[href$=".org" i] {
    font-style: italic;
}

/* <a> elements whose class attribute contains the word "logo" */
a[class~="logo"] {
    padding: 2px;
}
```

Pseudoclases

Una pseudoclase CSS es una palabra clave que se añade a los selectores y que especifica un estado especial del elemento seleccionado.

Enlace útil: https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes (listado de todas las pseudoclases)

:hover

La pseudo-clase :hover de CSS coincide cuando el usuario interactúa con un elemento con un dispositivo señalador, pero no necesariamente lo activa. Generalmente se activa cuando el usuario se desplaza sobre un elemento con el cursor (puntero del mouse).

```
.post button:hover {
    background-color: purple;
    color: white;
}

a:hover {
    text-decoration: underline;
}
```

:active

La pseudo-clase :active de CSS representa un elemento (como un botón) que el usuario está activando. Cuando se usa un mouse, la "activación" generalmente comienza cuando el usuario presiona el botón primario del mouse y termina cuando se suelta. La pseudo-clase :active se usa comúnmente en los elementos <a> y <button>, pero también se puede usar en otros elementos.

```
/* Selecciona cualquier <a> que esté siendo activado */
a:active {
    color: red;
}
```

Debes tener en cuenta que las pseudoclases ancla deben aparecer siempre en un determinado orden. Este orden es: **:link (no visitado), :visited (visitado), :hover y :active**. Por si te ayuda, para recordarlo, se emplean las iniciales: LVHA.

:nth-of-type

La pseudo-clase :nth-of-type() de CSS selecciona uno o más elementos de un tipo dado, en función de su posición entre un grupo de hermanos.

```
.post:nth-of-type(3) {
    background-color: seagreen;
     /*selecciona solo el tercero, pero si ponemos 3n, seleccionará cada
tres post*/
}
```

```
/* Selecciona cada cuarto elementos <p> entre
   cualquier grupo de hermanos */
   p:nth-of-type(4n) {
    color: lime;
  }
```

Existen muchas más pseudoclases, que se emplean menos, algunas de ellas:

- **:focus** hace referencia a los elementos que tienen el foco, como ocurre, por ejemplo, en los elementos de unformulario. Un ejemplo sería: input:focus {background-color: yellow;}
- **:first-child** hace referencia el primer hijo de un elemento padre. En el siguiente ejemplo se aplica el estilo alprimer elemento de una lista desordenada: ul li:first-child {font-weight: bold;}
- **:lang**(idioma) hace referencia al idioma en el que está un determinado elemento. En el siguiente ejemplo seaplica el estilo a cualquier párrafo que esté escrito en inglés:p:lang(en) {color:red}

Pseudoelementos

Al igual que las Pseudo-classes, los pseudo-elementos se añaden a los selectores, pero en cambio, no describen un estado especial, sino que, permiten añadir estilos a una parte concreta del documento. Por ejemplo, el pseudoelemento ::first-line selecciona solo la primera línea del elemento especificado por el selector.

Enlace útil: https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-elements

Sintaxis

```
selector::pseudo-elemento {
    propiedad: valor;
}
```

::after (:after)

En CSS, ::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.

```
/* Añdade una flecha después de los enlaces */
a::after {
    content: "→";
  }
```

::first-letter (:first-letter)

El pseudo-elemento ::first-letter aplica estilos a la primera letra de la primera línea un elemento de bloque, sólo cuando no es precedido de otro contenido (como imágenes o tablas).

```
h2::first-letter {
    font-size: 30px;
    color: black;
}
```

Especificidad en CSS y la "cascada"

El orden en el que se colocan los elementos en nuestra hoja de estilos importa. Como ya se ha podido intuir con anterioridad, en CSS, el concepto de "cascada" hace referencia a esta idea de que lo últimos elementos declarados tienen preferencia sobre los primeros en caso de conflicto. Por ejemplo, en el siguiente código:

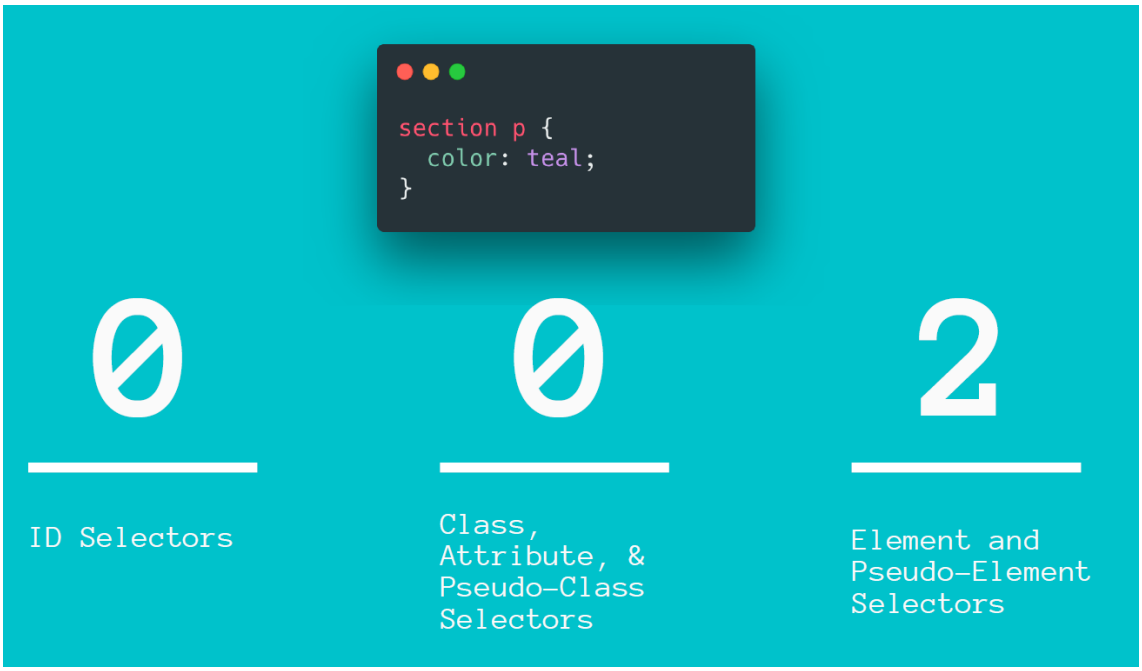
```
h1 {
    color: red;
}

h1 {
    color: blue;
}
```

El texto de los h1 va a ser de color azul.

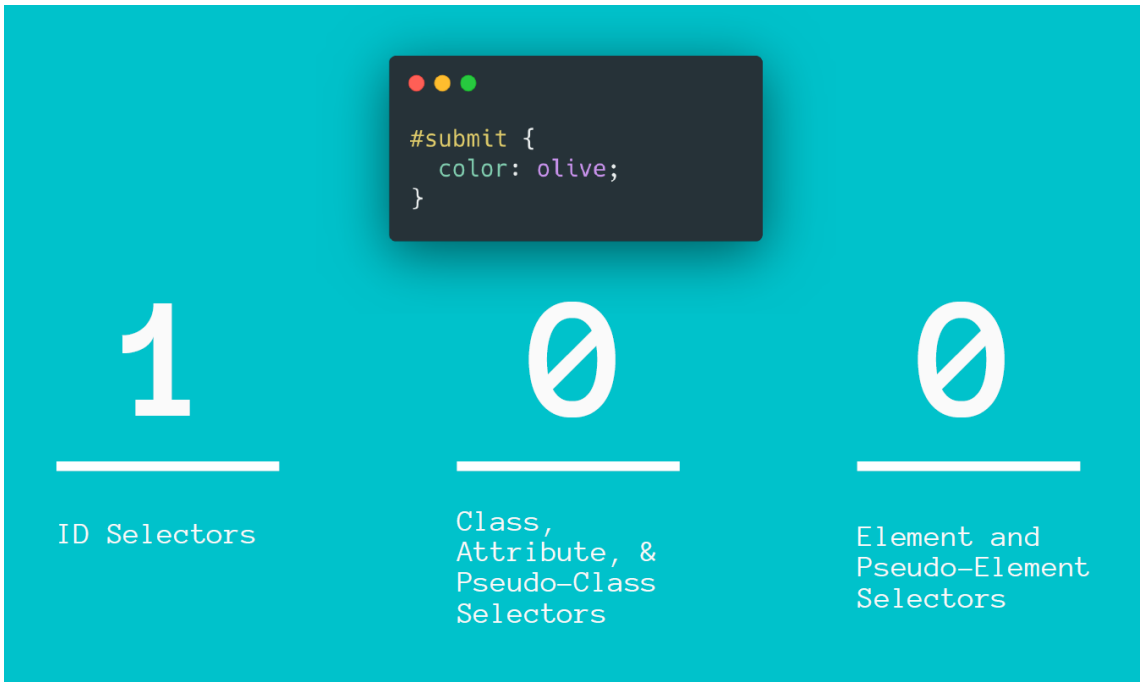
¿Qué es la especificidad?

Cuando el estilo aplicado a un mismo elemento entra en conflicto, la especificidad es la regla que siguen los navegadores para decidir que estilo es más "específico", de esta manera el estilo más específico es el que "gana" y finalmente se termina aplicando al elemento. Normalmente siguen la siguiente fórmula genérica de especificidad (aunque el cálculo exacto es más específico):

Es decir, un selector id es más específico que uno de clase, y este a su vez, más específico que el de elemento. El cálculo específico se realiza de la siguiente manera:



En el ejemplo anterior tenemos tres contadores por cada tipo de selectores en primer lugar los de ID, en segundo lugar: clase, atributo y pseudoclases y, por último: elementos y pseudoelementos. Según el tipo específico de selector que tenga cada regla de estilo se sumará 1 por cada uno en su lugar correspondiente. En el ejemplo hay dos selectores de elemento, que pertenecen a la tercera categoría, así que su puntuación es 002.



En esta imagen tenemos un selector ID, por lo que la puntuación será: 100.

Si ambas imágenes estuvieran haciendo referencia a un mismo elemento, ganaría el segundo, y ese es el color que se aplicaría “olive”.

Enlace útil: https://specificity.keegan.st/ (calculadora de especificidad)

Estilos en línea e !important (y por qué no usarlos)

Los estilos en línea son más específicos que los selectores ID. Así que, si algún estilo se está aplicando directamente sobre el elemento en el HTML, este será prioritario sobre los estilos que se estén aplicando en la hoja de estilos.

¡Important

Añadiendo la excepción “!important” al declarar un estilo, por ejemplo:

```
h1 {
    color: red !important;
}

h1 {
    color: blue;
}
```

Le estamos indicando al navegador que ese es el estilo que debe aplicar y, por tanto, que debe ignorar cualquier regla de especificidad y solo hacerle caso a esa línea (por eso se considera una excepción). Sin embargo, cuando dos declaraciones en conflicto con el !important son aplicadas al mismo elemento, se aplicará la declaración con mayor especificidad.

Es mejor evitar el uso de estilos en línea y de la excepción !important, su uso se considera una mala práctica y debería evitarse porque hace que el código sea más difícil de depurar al romper la cascada natural de las hojas de estilo.

Origen de la hoja de estilo

Los navegadores otorgan un peso distinto a las hojas de estilo que, ordenadas de menor a mayor peso, son:

- Hojas de estilo del navegador.
- Hojas de estilo del lector.
- Hojas de estilo de la persona que ha diseñado la página web.
- Declaraciones de estilo !important del lector.

Además de este orden, existe otra jerarquía de pesos que se aplican a las hojas de estilo creadas por la persona que ha diseñado la página web. Es importante entender esta jerarquía y tener en cuenta que las reglas de estilo que están al final de la lista ignorarán a las primeras. La siguiente lista, que como la anterior está ordenada de menor a mayor peso, muestra esta otra jerarquía:

- Hojas de estilo **externas vinculadas** (empleando el elemento **link** en la cabecera del documento).
- Hojas de estilo **externas importadas** (empleando el elemento **@import** dentro del elemento style en la cabecera del documento).
- Hojas de estilo **incrustadas** (empleando el elemento **style** en la cabecera del documento).
- **Estilos en línea** (empleando el **atributo style** en la etiqueta del elemento).
- Declaraciones de estilo marcadas como **!important**.

Herencia

Algunas propiedades CSS serán heredadas por elementos hijos, si éstas no son especificadas para el elemento hijo en concreto. Es decir, “heredarán” las propiedades de su elemento padre.

Por ejemplo, en el siguiente código:

HTML:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Css intro!</title>
    <link rel="stylesheet" href="try.css">
</head>

<body>
    <h1>¡Hola mundo!</h1>
    <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam
cum veritatis rerum, itaque, consectetur illum
          porro, eos adipisci accusamus tenetur facilis dolorem quod
doloremque eius voluptatem aliquam voluptatibus
        laudantium voluptate?</p>
</body>

</html>
```

CSS:

```css
body {
    color: purple;
}
```

El resultado visual final en nuestra página será:

No hemos especificado ninguna propiedad en el elemento <h1> ni el elemento <p>, sin embargo, sí hemos especificado el color del elemento <body> (padre de los dos anteriores), por lo que estos últimos han heredado el color violeta de su elemento padre.

Algunos elementos, sin embargo, no heredarán los estilos aplicados a su elemento padre, a menos que lo especifiquemos así. (Además, también existen propiedades heredables y no heredables). Por ejemplo, en el siguiente código:

HTML:

```html
<form action="">
        <label for="username">Username:</label>
        <input type="text" placeholder="username" id="username">

        <button>Send</button>
</form>
```

CSS:

```css
body {
    color: purple;
}

form {
    color: red;
}
```

El resultado es:

¡Hola mundo!

Lorem ipsum dolor sit amet consectetur adipisicing e

Username: username Send

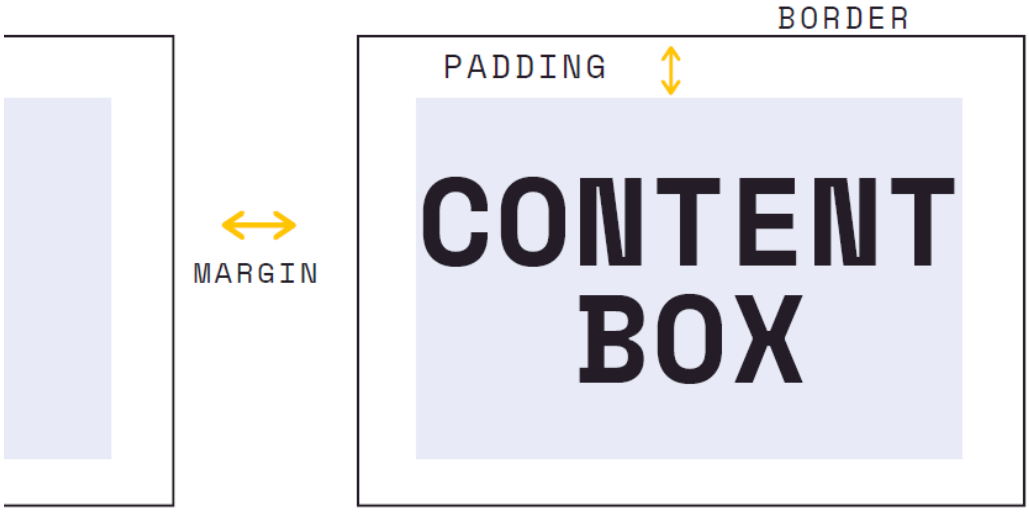
Vemos que el color rojo se aplica al elemento <label> pero no al <input> ni al <button>. Si queremos que estos elementos hereden el color del elemento padre <form> como lo hizo <label>, debemos especificarlo en la hoja de estilos de la siguiente manera:

```
input,
button {
    color: inherit;
}
```

El modelo de caja en CSS (CSS Box Model)

Todo en CSS tiene una caja alrededor y comprender estas cajas es clave para poder crear diseños con CSS o para alinear elementos con otros elementos.

The Box Model



De acuerdo con este modelo, todos los elementos de una página web generan una caja rectangular alrededor llamada **caja del elemento**.

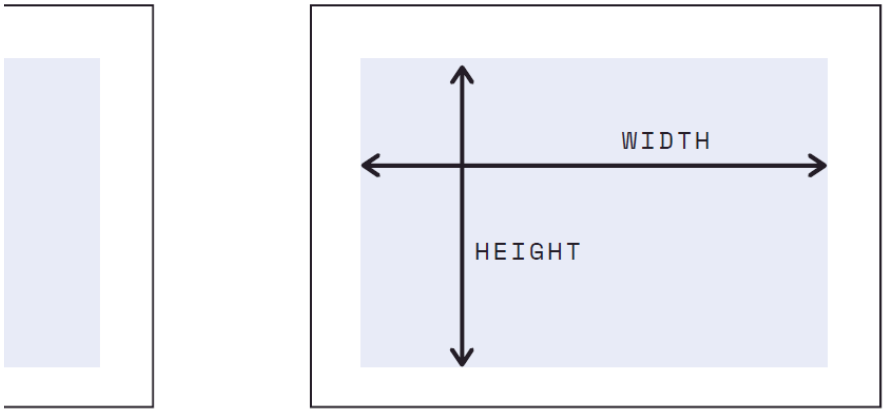
El modelo de caja en CSS cuenta con los elementos siguientes:

El contenido de la caja (o content box)

El área donde se muestra el contenido, cuyo tamaño puede cambiarse utilizando propiedades como **width y height.**

Las propiedades CSS width y height establecen el ancho y alto de un elemento. (Nota: Por defecto, se establece el ancho y alto del **content box**, pero si el **box-sizing** se establece en **border-box**, establece el ancho del **border box**.)

Width & Height



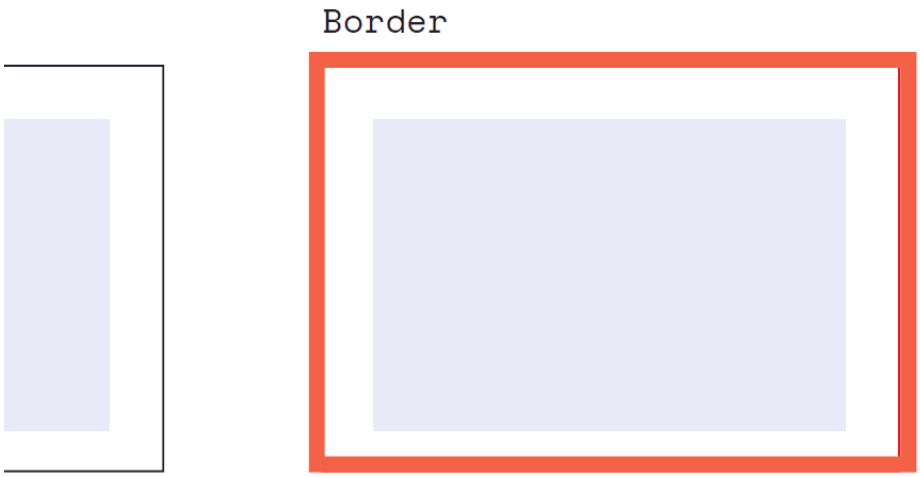
Ejemplo y sintaxis

```
h1 {
    width: 250px;
    height: 100px;
}
```

El borde de la caja (o border box)

El borde de la caja envuelve el contenido y el relleno. Es posible controlar su tamaño y estilo utilizando la propiedad **border** y otros**.**

La propiedad border permite definir en una única regla todos los bordes de los elementos seleccionados. Se puede utilizar border para definir el o los valores siguientes: border-width, border-style, border-color.



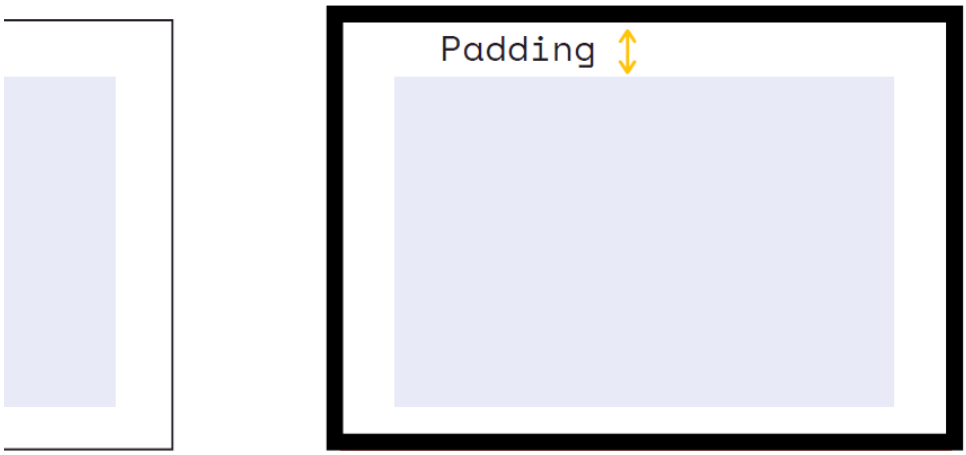
Ejemplo y sintaxis

```
/* style */
border: solid;

/* width | style */
border: 2px dotted;

/* style | color */
border: outset #f33;

/* width | style | color */
border: medium dashed green;
```

También se puede estilizar solo uno de los lados del borde, para ello se cuentan con propiedades como border-top-width, border-right-style o border-bottom-color, etc. Siguiendo esta nomenclatura: border-(top/right/bottom/left)-propiedad. También existe la propiedad **border-radius**, que nos permite "redondear" los vértices del borde de la caja.

El relleno de la caja (o padding box)

El relleno es espacio en blanco alrededor del contenido; es posible controlar su tamaño usando la propiedad **padding** (padding-top, padding-right, padding-bottom, padding-left).



Ejemplo y sintaxis

```
/* Apply to all four sides */
padding: 1em;

/* top and bottom | left and right */
padding: 5% 10%;

/* top | left and right | bottom */
padding: 1em 2em 2em;

/* top | right | bottom | left */
padding: 5px 1em 0 2em;
```

El margen de la caja (o margin box)

El margen es la capa más externa. Envuelve el contenido, el relleno y el borde como espacio en blanco entre la caja y otros elementos. Es posible controlar su tamaño usando la propiedad **margin.** Su uso es similar al de padding.

Los márgenes **top y bottom** de dos elementos que van seguidos se "**colapsan**". Es decir, se asume como margen entre ambos elementos el mayor de ellos. Por el contrario, si fuesen dos elementos colocados "**uno al lado del otro**" (como dos elementos span), deberíamos tener en cuenta que los márgenes **right y left** no se colapsan, sino que se suman.

Hay algunas características fundamentales del modelo de cajas que vale la pena destacar:

- El relleno, los bordes y los márgenes son opcionales, por lo que, si ajustas a cero sus valores se eliminarán de la caja.
- Cualquier color o imagen que apliques de fondo al elemento se extenderá por el relleno.
- Los bordes se generan con propiedades de estilo que especifican su estilo (por ejemplo: sólido), grosor y color. Cuando el borde tiene huecos, el color o imagen de fondo aparecerá a través de esos huecos.
- Los márgenes siempre son transparentes (el color del elemento padre se verá a través de ellos).
- Cuando definas el largo de un elemento estás definiendo el largo del área de contenido (los largos de relleno, de borde y de márgenes se sumarían a esta cantidad).
- Puedes cambiar el estilo de los lados superior, derecho, inferior e izquierdo de una caja de un elemento por separado.

Cajas en bloque y en línea (propiedad display)

En CSS, en general, hay dos tipos de cajas: cajas en bloque y cajas en línea (esto ya se vio en HTML cuando hablábamos de elementos de línea o en línea y elementos de bloque). Estas características se refieren a cómo se comporta la caja en términos de flujo de página y en relación con otras cajas de la página:

Si una caja se define como un bloque, se comportará de las maneras siguientes:

- La caja fuerza un salto de línea al llegar al final de la línea.
- La caja se extenderá en la dirección de la línea para llenar todo el espacio disponible que haya en su contenedor. En la mayoría de los casos, esto significa que la caja será tan ancha como su contenedor, y llenará el 100% del espacio disponible.
- Se respetan las propiedades width y height.
- El relleno, el margen y el borde mantienen a los otros elementos alejados de la caja.

Si una caja tiene una visualización externa de tipo inline, entonces:

- La caja no fuerza ningún salto de línea al llegar al final de la línea.
- Las propiedades width y height no se aplican.
- Se aplican relleno, margen y bordes verticales, pero no mantienen alejadas otras cajas en línea (eje vertical no se respeta).
- Se aplican relleno, margen y bordes horizontales, y mantienen alejadas otras cajas en línea (eje horizontal se respeta).

Visualización inline-block: se comporta como una caja con visualización de tipo inline, excepto que:

- Las propiedades width y height se aplican.
- Las propiedades margin y padding se respetan.

El tipo de caja que se aplica a un elemento está definido por los valores de propiedad **display**, como **block** e **inline**, o **inline-block**. (Hay otras opciones que veremos más adelante al hablar de Flexbox y el grid).

Ejemplo

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Css intro!</title>
    <link rel="stylesheet" href="try.css">
</head>

<body>
    <div></div>
    <div></div>
    <div></div>
</body>

</html>
```

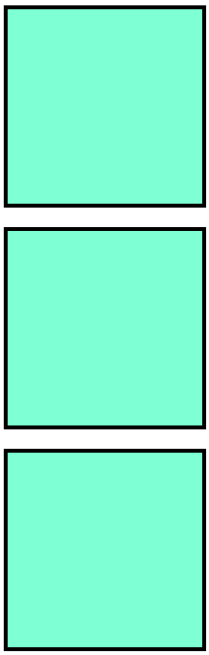
CSS

```
div {
```

```
    width: 250px;
    height: 250px;
    background-color: aquamarine;
    margin: 25px;
    border: 5px solid black;
}
```

*El elemento div por defecto es un elemento de bloque.

Navegador:

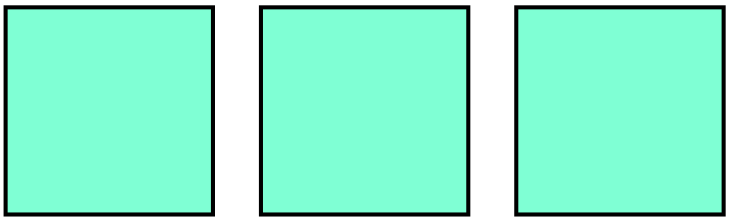


```
div {
    width: 250px;
    height: 250px;
    background-color: aquamarine;
    margin: 25px;
    border: 5px solid black;
    display: inline;
}
```

Navegador:

```
div {
    width: 250px;
    height: 250px;
    background-color: aquamarine;
    margin: 25px;
    border: 5px solid black;
    display: inline-block;
}
```

Navegador:



Unidades en CSS

Unidades absolutas

Todas las unidades siguientes son unidades de longitud absoluta: no son relativas a nada más y en general se considera que siempre tienen el mismo tamaño.

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 = 1/6 de 1in
`pt`	Puntos	1pt = 1/72 de 1in
`px`	Píxeles	1px = 1/96 de 1in

La unidad más común con diferencia es el píxel.

Unidades 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 (diseño responsive). En la tabla siguiente se enumeran algunas de las unidades más útiles para el desarrollo web.

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`.
`ex`	Altura x de la fuente del elemento.
`ch`	La medida de avance (ancho) del glifo "0" de la letra del elemento.
`rem`	Tamaño de la letra del elemento raíz.
`lh`	Altura de la línea del elemento.
`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.

Las unidades **em** y **rem** son de las más habituales.

En esta tabla también podríamos añadir **los porcentajes**. Los porcentajes pueden ser relativos a un elemento padre o al propio elemento (normalmente respecto a su font-size).

Nota: Si tenemos muchos elementos anidados y trabajamos con unidades em, debido a que la propiedad font-size es relativa al elemento padre, podemos estar aumentando (o reduciendo) de manera geométrica los tamaños de las fuentes. En estos casos es mejor usar unidades rem.

Por otra parte, las unidades em son muy útiles para estilizar cosas como los bordes de un botón, ya que las proporciones del botón se mantendrán, aunque cambiemos su tamaño.

Otras propiedades útiles en CSS

Posicionamiento

Un término que aparecerá a menudo es “flujo normal”. Cuando hablamos de que los objetos de una página siguen el flujo normal del documento, queremos indicar que la forma en la que se disponen en la ventana del navegador coincide con el lugar que ocupan en el documento escrito (en el código(X)HTML), donde el orden de lectura es de arriba a abajo y de izquierda a derecha.

Flotando y posicionando con CSS conseguimos que los elementos abandonen su flujo normal. De esta forma un elemento que este en el documento escrito más abajo que otro en el documento puede verse en el navegador por encima de él.

display

Esta propiedad permite al documento interpretar de otra forma los elementos de tipo bloque y los elementos de tipo línea. Para ello basta con que asignes a esta propiedad el valor block si quieres que un elemento "en línea" se comporte como un elemento de tipo bloque y none, si

quieres que un elemento de bloque no genere caja, no muestre su contenido y no ocupe espacio en la página.

position

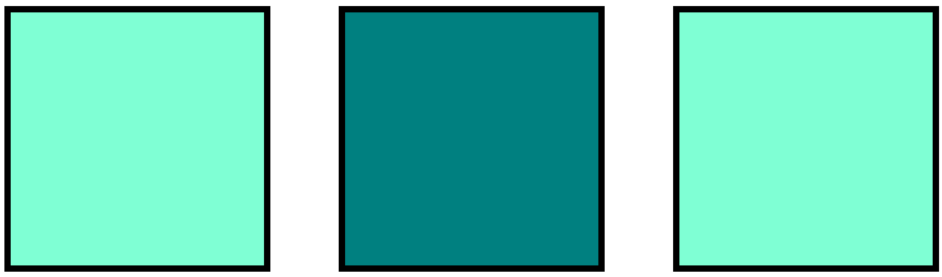
La **propiedad position** de CSS especifica cómo un elemento es posicionado en el documento. Las propiedades **top, right, bottom** y **left** determinan la ubicación final de los elementos posicionados.

Posibles valores:

Static

El elemento se posiciona acorde a flujo normal del documento. Es el valor por defecto. Top, left, right, bottom y z-index no tienen efecto.

```
#dos {/*El estilo se aplica al Segundo cuadrado*/
    background-color: teal;
    position: static;
}
```

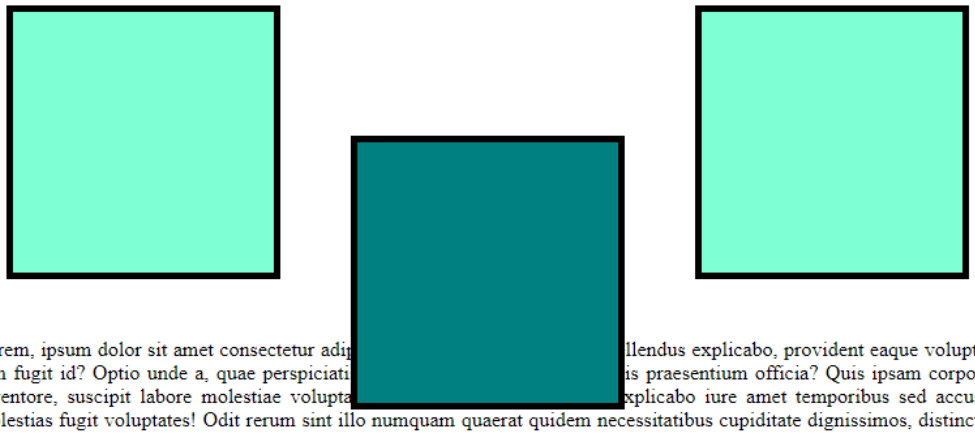


Lorem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit eos u
rem fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magni max
inventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupiditate fu
molestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem minus. (
facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo aliq
necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, voluptatil
Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laborio
ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam n
perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta facere
animi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur voluptati
Asperiores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non accusar
optio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit molestias
voluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum sapie
assumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur numq
deleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati! Quo
quibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero odit p
sequi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

Relative

El elemento se coloca de acuerdo con el flujo normal del documento y luego se desplaza con respecto a sí mismo según los valores de top, right, bottom y left. El desplazamiento no afecta la posición de ningún otro elemento; por lo tanto, el espacio dado para el elemento en el diseño de página es el mismo que si la posición fuera estática.

```
#dos {
    background-color: teal;
    position: relative;
    top: 100px;
}
```



.orem, ipsum dolor sit amet consectetur adi llendus explicabo, provident eaque voluptate totam odit eo em fugit id? Optio unde a, quae perspiciati is praesentium officia? Quis ipsam corporis modi magni r nventore, suscipit labore molestiae volupta xplicabo iure amet temporibus sed accusamus, cupiditate nolestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem minu acilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo ecessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, volupt nimi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, labo llam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum tota erferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta face nimi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur volup speriores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non accu ptio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit molesti oluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum s ssumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur nu eleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati! C uibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero od equi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

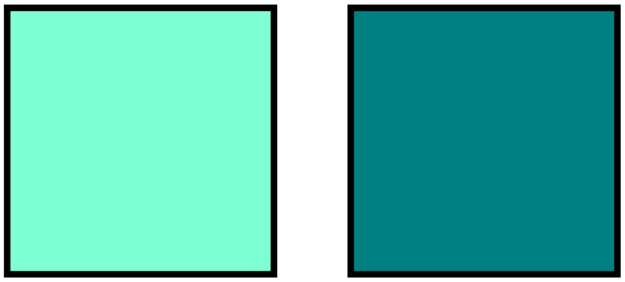
Un elemento posicionado relativamente, que siga en el flujo normal del XHTML inmediatamente después a otro elemento posicionado también relativamente, calculará su origen de la forma siguiente:

- Si el elemento es hijo del anterior, su origen estará en el final del anterior (su padre).
- Si el elemento no es hijo del anterior, tendrá su origen donde el anterior tenga su final si no se fijaron valores distintos de cero en sus propiedades top y left.

Absolute

El elemento se elimina del flujo normal del documento y no se crea ningún espacio para el elemento en el diseño de página. El elemento se posiciona en relación con su ancestro posicionado más cercano (si lo hay) o con respecto al bloque contenedor inicial. Su posición final está determinada por los valores de top, right, bottom y left.

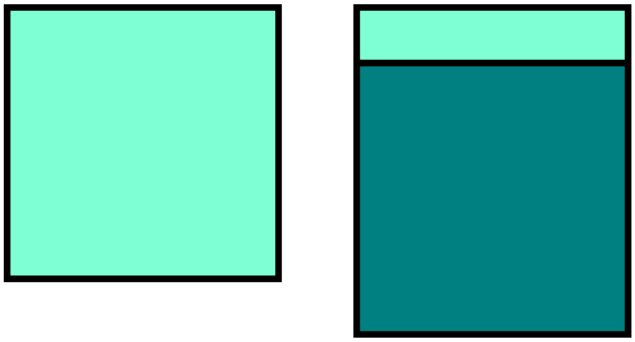
```
#dos {
    background-color: teal;
    position: absolute;
}
```



Lorem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit ec
rem fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magni
inventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupiditat
molestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem min
facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo
necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, volup
Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, lab
ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum tota
perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta fac
animi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur volup
Asperiores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non acc
optio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit molest
voluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum s
assumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur nu
deleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati! (
quibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero od
sequi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

Al haber “eliminado” el segundo cuadrado del flujo del documento, lo que ha sucedido es que el cuadrado tres ha ocupado su lugar (estaría en la segunda posición). La razón por la que vemos en ese lugar al segundo elemento es porque este se ha posicionado absolutamente en esa posición (al margen del flujo del documento) así que lo que realmente estamos viendo es al cuadrado dos sobre el tres.

Si movemos el cuadrado dos podremos comprobar que el tres está debajo.



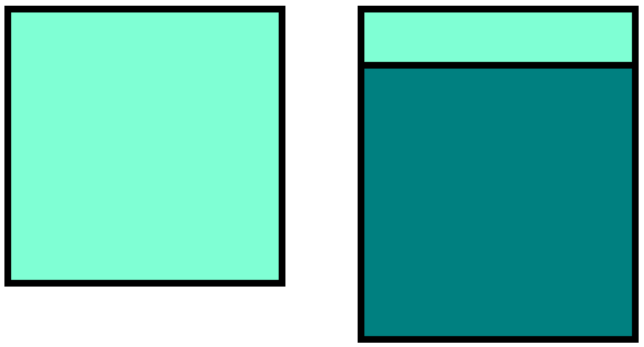
.orem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit
em fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magn
nventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupidit
nolestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem mi
acilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo
ecessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, volu
nimi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, la
llam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum to
erferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta fa
nimi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur vol
speriores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non ac
ptio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit mole
oluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum
ssumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur
eleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati!
uibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero
equi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

Además, su nueva posición es relativa al bloque contenedor inicial (body), ya que no se encuentra dentro de ningún otro elemento posicionado (con un valor de posición distinto de static).

Fixed

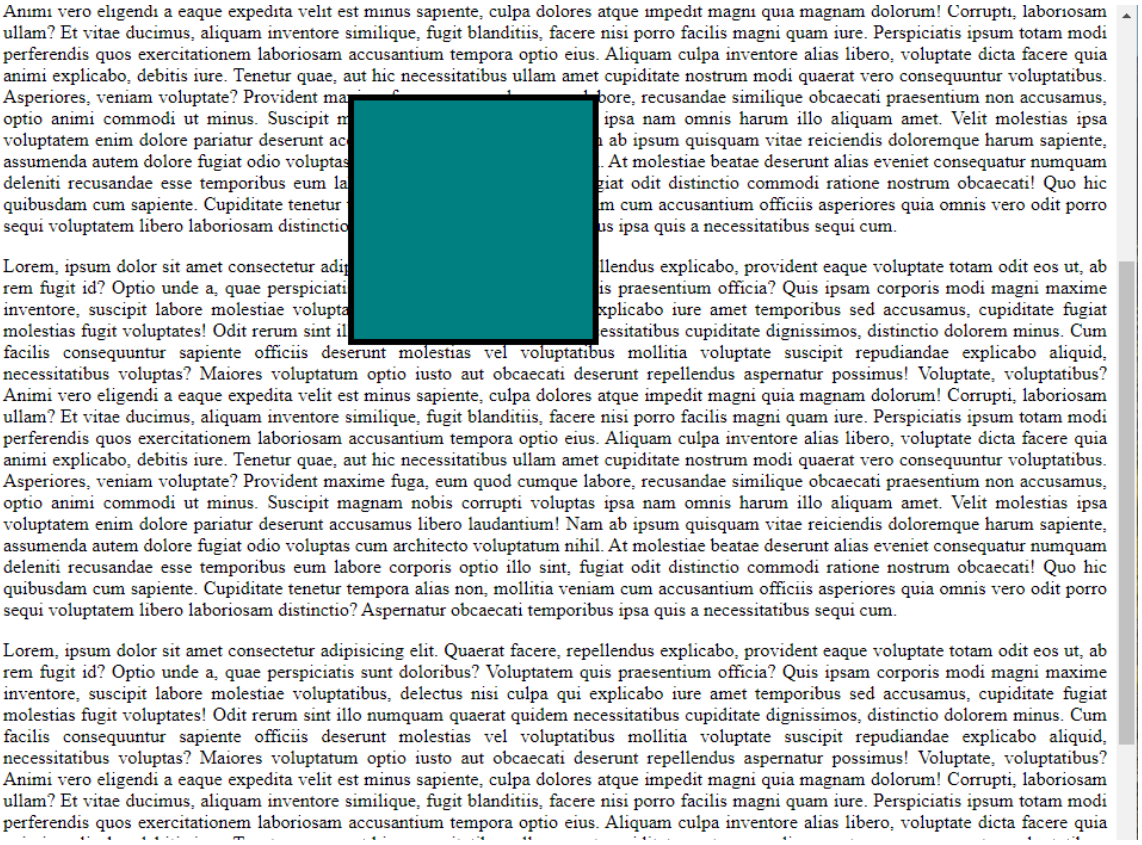
El elemento se elimina del flujo normal del documento y no se crea ningún espacio para el elemento en el diseño de página. El elemento se coloca en relación con su bloque contenedor inicial, que es la ventana gráfica en el caso de medios visuales. Su posición final está determinada por los valores de top, right, bottom y left. Este valor siempre crea un nuevo contexto de apilamiento. En los documentos impresos, el elemento se coloca en la misma posición en cada página.

```
#dos {
    background-color: teal;
    position: fixed;
    top: 50px;

}
```



.orem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odi
em fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi mag
nventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupid
nolestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem n
acilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicab
ecessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, vo
Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti,
illam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum t
erferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta
nimi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur vo
Asperiores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non a
optio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit mol
voluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harur
ssumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur
leleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecat
quibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero
equi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

Parece que no cambia con respecto al caso de absolute, pero si hacemos scroll down:



Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laboriosam ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam modi perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta facere quia animi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur voluptatibus. Asperiores, veniam voluptate? Provident ma bore, recusandae similique obcaecati praesentium non accusamus, optio animi commodi ut minus. Suscipit m ipsa nam omnis harum illo aliquam amet. Velit molestias ipsa voluptatem enim dolore pariatur deserunt ac ab ipsum quisquam vitae reiciendis doloremque harum sapiente, assumenda autem dolore fugiat odio voluptas . At molestiae beatae deserunt alias eveniet consequatur numquam deleniti recusandae esse temporibus eum la giat odit distinctio commodi ratione nostrum obcaecati! Quo hic quibusdam cum sapiente. Cupiditate tenetur m cum accusantium officiis asperiores quia omnis vero odit porro sequi voluptatem libero laboriosam distinctio us ipsa quis a necessitatibus sequi cum.

Lorem, ipsum dolor sit amet consectetur adi llendus explicabo, provident eaque voluptate totam odit eos ut, ab rem fugit id? Optio unde a, quae perspiciati is praesentium officia? Quis ipsam corporis modi magni maxime inventore, suscipit labore molestiae volupta xplicabo iure amet temporibus sed accusamus, cupiditate fugiat molestias fugit voluptates! Odit rerum sint il essitatibus cupiditate dignissimos, distinctio dolorem minus. Cum facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo aliquid, necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, voluptatibus? Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laboriosam ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam modi perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta facere quia animi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur voluptatibus. Asperiores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non accusamus, optio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit molestias ipsa voluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum sapiente, assumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur numquam deleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati! Quo hic quibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero odit porro sequi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit eos ut, ab rem fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magni maxime inventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupiditate fugiat molestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem minus. Cum facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo aliquid, necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, voluptatibus? Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laboriosam ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam modi perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta facere quia

Vemos que el elemento se ha quedado “fijado” en la misma posición respecto al marco del navegador.

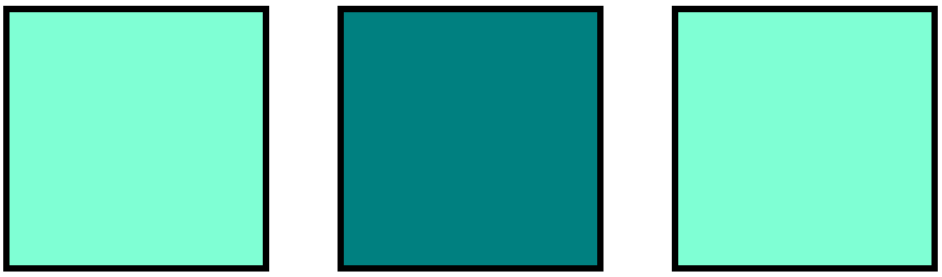
Sticky

El elemento es posicionado de acuerdo al flujo normal del documento y luego se desplaza respecto a su padre y bloque contenedor, según los valores top, right, left y bottom. El desplazamiento no afecta a otros elementos.

Se puede entender como que el elemento con la propiedad “sticky” se “pega” a su elemento padre, según la posición que determinemos, cuando hacemos scroll down.

```
#dos {
    background-color: teal;
    position: sticky;
    top: 50px;
}
```

Lorem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit eos u
rem fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magni ma
inventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupiditate fi
molestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem minus.
facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo ali
necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, voluptati
Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laborio
ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam
perferendis quos exercitationem laboriosam accusantium tempora optio eius. Aliquam culpa inventore alias libero, voluptate dicta facere
animi explicabo, debitis iure. Tenetur quae, aut hic necessitatibus ullam amet cupiditate nostrum modi quaerat vero consequuntur voluptat
Asperiores, veniam voluptate? Provident maxime fuga, eum quod cumque labore, recusandae similique obcaecati praesentium non accusa
optio animi commodi ut minus. Suscipit magnam nobis corrupti voluptas ipsa nam omnis harum illo aliquam amet. Velit molestias
voluptatem enim dolore pariatur deserunt accusamus libero laudantium! Nam ab ipsum quisquam vitae reiciendis doloremque harum sapi
assumenda autem dolore fugiat odio voluptas cum architecto voluptatum nihil. At molestiae beatae deserunt alias eveniet consequatur numq
deleniti recusandae esse temporibus eum labore corporis optio illo sint, fugiat odit distinctio commodi ratione nostrum obcaecati! Quo
quibusdam cum sapiente. Cupiditate tenetur tempora alias non, mollitia veniam cum accusantium officiis asperiores quia omnis vero odit p
sequi voluptatem libero laboriosam distinctio? Aspernatur obcaecati temporibus ipsa quis a necessitatibus sequi cum.



Lorem, ipsum dolor sit amet consectetur adipisicing elit. Quaerat facere, repellendus explicabo, provident eaque voluptate totam odit eos u
rem fugit id? Optio unde a, quae perspiciatis sunt doloribus? Voluptatem quis praesentium officia? Quis ipsam corporis modi magni ma
inventore, suscipit labore molestiae voluptatibus, delectus nisi culpa qui explicabo iure amet temporibus sed accusamus, cupiditate fi
molestias fugit voluptates! Odit rerum sint illo numquam quaerat quidem necessitatibus cupiditate dignissimos, distinctio dolorem minus.
facilis consequuntur sapiente officiis deserunt molestias vel voluptatibus mollitia voluptate suscipit repudiandae explicabo ali
necessitatibus voluptas? Maiores voluptatum optio iusto aut obcaecati deserunt repellendus aspernatur possimus! Voluptate, voluptati
Animi vero eligendi a eaque expedita velit est minus sapiente, culpa dolores atque impedit magni quia magnam dolorum! Corrupti, laborio
ullam? Et vitae ducimus, aliquam inventore similique, fugit blanditiis, facere nisi porro facilis magni quam iure. Perspiciatis ipsum totam

Hacemos scroll down y cuando alcanzamos el elemento:

Vemos como se queda “pegado” o “fijado” en la posición que hemos indicado con la propiedad top (en este caso, relativa al “marco” del navegador ya que no tienen ningún elemento padre posicionado).

Float

Flotar sirve para mover una caja a la izquierda o a la derecha hasta que su borde exterior toque el borde de la caja que lo contiene o toque otra caja flotante. Para que un elemento pueda flotar debe tener definido implícita o explícitamente su tamaño.

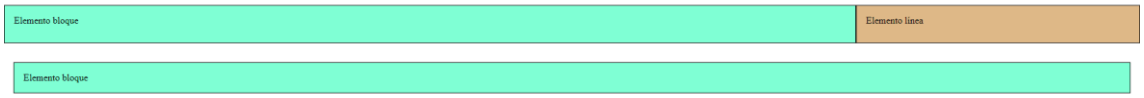
La propiedad **float** elimina los elementos de su flujo normal en el documento y los coloca a la izquierda o la derecha de su elemento padre, permitiendo a los elementos de texto y en línea aparecer a su costado. El elemento es removido del normal flujo de la página, aunque aún sigue siendo parte del flujo (a diferencia del posicionamiento absoluto). Posibles valores**: none, left, right** e **inherit.**

```
    float: left;
```

Ejemplo:

```
#primer-div {
    width: 75%;
    height: 4rem;
    box-sizing: border-box;
    margin: 1rem 0rem;
    display: inline-block;
```

```
}

#primer-span {
    height: 4rem;
    width: 25%;
    box-sizing: border-box;
    margin: 1rem 0rem;
    float: right;
}
```



Al establecer la propiedad float con algún valor distinto de none, el elemento pasa a comportarse como un elemento de bloque (se respetan las propiedades width y height).

Clear

Sirve para mantener limpia el área que está al lado del elemento flotante y que el siguiente elemento comience en su posición normal dentro del bloque que lo contiene. Valores: **left, right, both, none** e **inherit.**

visibility

Esta propiedad controla si el elemento será visualizado según le asignes el valor visible o hidden. Debes tener en cuenta que, aunque un elemento no sea visible, éste continúa ocupando su espacio en el flujo normal del documento al contrario de lo que ocurría con la propiedad display cuando se le asignaba el valor none.

z-index

Especifica el orden en el que los elementos se apilan encima unos de otros. Debe ser un entero. Los valores más altos son para los elementos que se encuentren más "encima". Es decir, un elemento con z-index 3 estará por encima de un elemento con z-indez 2.

Por defecto, los elementos se apilan en el orden en que aparecen: el elemento situado más abajo en el flujo normal del documento quedará encima. Si quieres que esta posición no sea tenida en cuenta, debes saber que los elementos con un valor mayor de la propiedad z-index son colocados encima de los que tienen un valor menor z-index, quedando estos últimos tapados por los primeros.

También debes saber que esta propiedad sólo se aplica a elementos que tengan la propiedad position absolute o en relative.

Transiciones

La propiedad **transition** es una propiedad abreviada de transition-property, transition-duration, transition-timing-function, y transition-delay. (Permite definir la transición entre dos

estados de un elemento. Hay diferentes estados que pueden ser definidos utilizando pseudo-clases como :hover o :active o aplicado dinámicamente usando JavaScript.

Transition-delay

Especifica la cantidad de tiempo a esperar entre un cambio pedido hacia una propiedad y el comienzo de un efecto de transición. Se mide en segundos (s) o milisegundos (ms).

El retraso puede ser cero, positivo o negativo:

- Un valor de 0 s (o 0 ms) comenzará el efecto de transición inmediatamente.
- Un valor positivo retrasará el inicio del efecto de transición durante el período de tiempo determinado.
- Un valor negativo iniciará el efecto de transición inmediatamente y durante la mitad del efecto. En otras palabras, el efecto se animará como si ya hubiera estado ejecutándose durante el período de tiempo indicado.

```
/* <time> values */
transition-delay: 3s;
transition-delay: 2s, 4ms;
```

Se pueden especificar distintos tiempos, cada uno se aplicará a una propiedad distinta especificada en:

Transition-property

La propiedad de transición establece las propiedades CSS a las que se debe aplicar un efecto de transición. Además, acepta los valores “none” y “all”.

Transition-duration

La propiedad CSS de duración de transición establece el tiempo que debe tardar en completarse una animación de transición. De forma predeterminada, el valor es 0, lo que significa que no se producirá ninguna animación.

Transition-timing-function

La propiedad CSS de función de sincronización de transición establece cómo se calculan los valores intermedios para las propiedades CSS que se ven afectadas por un efecto de transición.

Esto, en esencia, permite establecer una curva de aceleración para que la velocidad de la transición pueda variar a lo largo de su duración.

Esta curva de aceleración se define utilizando una <función de aceleración> para cada propiedad a la que se realizará la transición.

Puede especificar múltiples funciones de aceleración; cada uno se aplicará a la propiedad correspondiente según lo especificado por la propiedad de propiedad de transición, que actúa como una lista de propiedades de transición. Si se especifican menos funciones de aceleración que en la lista de propiedades de transición, el agente de usuario debe calcular qué valor se utiliza repitiendo la lista de valores hasta que haya uno para cada propiedad de transición. Si hay más funciones de facilitación, la lista se trunca al tamaño correcto. En ambos casos, la declaración CSS sigue siendo válida.

Enlaces de interés: https://easings.net/

Transition (ejemplos de uso y sintaxis)

```
/* Apply to 1 property */
/* property name | duration */
transition: margin-right 4s;

/* property name | duration | delay */
transition: margin-right 4s 1s;

/* property name | duration | easing function */
transition: margin-right 4s ease-in-out;

/* property name | duration | easing function | delay */
transition: margin-right 4s ease-in-out 1s;

/* Apply to 2 properties */
transition:
  margin-right 4s,
  color 1s;

/* Apply to all changed properties */
transition: all 0.5s ease-out;
```

Transformaciones

La propiedad CSS **transform** permite rotar, escalar, sesgar o trasladar un elemento. Modifica el espacio de coordenadas del modelo de formato visual CSS. La propiedad de transformación se puede especificar como el valor de la palabra clave “none” o como uno o más valores de <transform-function>.

Ejemplos

```
/* Keyword values */
transform: none;

/* Function values */
transform: matrix(1, 2, 3, 4, 5, 6);
transform: matrix3d(1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1);
transform: perspective(17px);
transform: rotate(0.5turn);
transform: rotate3d(1, 2, 3, 10deg);
transform: rotateX(10deg);
transform: rotateY(10deg);
transform: rotateZ(10deg);
transform: translate(12px, 50%);
transform: translate3d(12px, 50%, 3em);
transform: translateX(2em);
transform: translateY(3in);
transform: translateZ(2px);
transform: scale(2, 0.5);
transform: scale3d(2.5, 1.2, 0.3);
transform: scaleX(2);
```

```
transform: scaleY(0.5);
transform: scaleZ(0.3);
transform: skew(30deg, 20deg);
transform: skewX(30deg);
transform: skewY(1.07rad);
/* Multiple function values */
transform: translateX(10px) rotate(10deg) translateY(5px);
transform: perspective(500px) translate(10px, 0, 20px) rotateY(3deg);
```

box-shadow

La propiedad CSS **box-shadow** agrega efectos de sombra alrededor del marco de un elemento. Puede configurar múltiples efectos separados por comas. La sombra de un cuadro se describe mediante desplazamientos X e Y en relación con el elemento, radio de desenfoque y extensión y color.

Ejemplos

```
/* Keyword values */
box-shadow: none;

/* offset-x | offset-y | color */
box-shadow: 60px -16px teal;

/* offset-x | offset-y | blur-radius | color */
box-shadow: 10px 5px 5px black;

/* offset-x | offset-y | blur-radius | spread-radius | color */
box-shadow: 2px 2px 2px 1px rgba(0, 0, 0, 0.2);

/* inset | offset-x | offset-y | color */
box-shadow: inset 5em 1em gold;

/* Any number of shadows, separated by commas */
box-shadow:
  3px 3px red,
  -1em 0 0.4em olive;
```

background

La propiedad **background** es un atajo para definir los valores individuales del fondo en una única regla CSS. Se puede usar background para definir los valores de una o de todas las propiedades siguientes: **background-attachment, color, image, position, repeat.**

Ejemplos

```
/* Using a <background-color> */
background: green;

/* Using a <bg-image> and <repeat-style> */
background: url("test.jpg") repeat-y;

/* Using a <box> and <background-color> */
background: border-box red;
```

No tiene ninguna propiedad obligatoria y sus valores pueden aparecer en cualquier orden. Sólo debes tener una restricción: la posición se indica con dos valores que deben aparecer juntos, primero el horizontal seguido inmediatamente después por el vertical, ya que si sólo aplicamos un valor el otro se configura por defecto a center.

```
/* A single image, centered and scaled */
background: no-repeat center/80% url("../img/image.png");
```

Background-image

Esta propiedad sirve para configurar la imagen de fondo de cualquier elemento. La propiedad **background-image** prevalece sobre la propiedad **background-color** por lo que si con un elemento realizas declaraciones con estas dos propiedades ignorará la declaración de background-color.

```
background-image: url("test.jpg");
```

Background-repeat

Permite configurar a tu gusto la forma en la que se repetirá la imagen cuando su tamaño sea más pequeño que la ventana del navegador. También permite evitar que la imagen se repita.

```
background-repeat: no-repeat;
background-repeat: repeat-x;
background-repeat: repeat-y;
```

Background-position

Especifica la posición de la primera imagen que cubrirá el fondo del elemento en el que esté definida esta propiedad. Al posicionamiento podemos asignarles los valores: **left, right, top, bottom** y **center** los cuales se usan, normalmente, de dos en dos sin importar el orden (uno indica su posición horizontal y el otro indica su posición vertical). Si sólo se indica una, se supone que la otra es center.

```
background-position: top center;
```

Para el posicionamiento se pueden emplear también las medidas de longitud absolutas o relativas. En este caso, las medidas son relativas al extremo superior izquierdo del elemento.

```
background-position: 150px 150px;
```

También puedes utilizar los valores porcentuales. Como seguro supones, si indicas sólo un valor se asume que el otro es un 50%. Debes tener en cuenta que el valor porcentual se aplica “al contenedor y a la imagen en sí”.

```
background-position: 50% 50%;
```

background-attachment

Con esta propiedad puedes fijar la imagen en una posición concreta. Se le pueden asignar los valores**: scroll, fixed** e **inherit,** siendo scroll el valor por defecto. Se suele emplear el valor fixed para conseguir que la imagen no se desplace con el documento.

Google Fonts

Enlace de interés: https://fonts.google.com/

Cuando escogemos una fuente para estilizar un documento, y nos queremos asegurar de que la misma se va a poder ver en cualquier navegador, podemos incluir manualmente dicha fuente en el código. Esto lo haremos normalmente mediante la etiqueta HTML <link>. Por ejemplo:

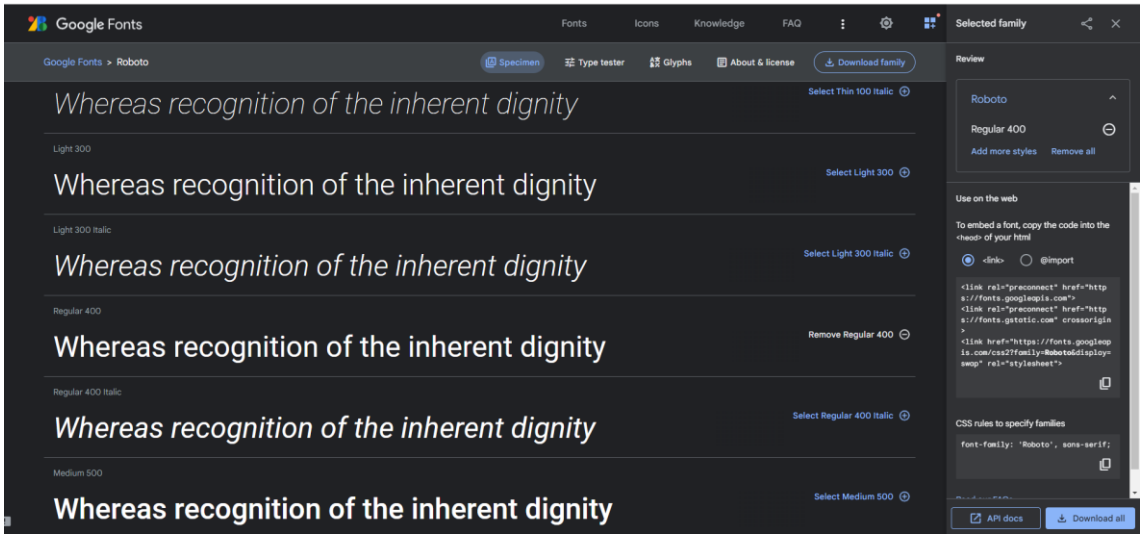
Para importar una fuente gratuita de Google Fonts:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>CSS Practice</title>
    <link rel="stylesheet" href="try.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
                                                                <link
href="https://fonts.googleapis.com/css2?family=Roboto&display=swap"
rel="stylesheet">
</head>

<body>
    <h1>Google Fonts</h1>
    <p>Probando la fuente Roboto, de Google Fonts.</p>

</body>

</html>
```

Y en el documento CSS:

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

El código que debemos añadir y dónde nos lo proporciona la propia página de Google Fonts.



Crear una variable CSS customizada

Para crear una variable en CSS, tan solo necesitamos darle un nombre siguiendo la siguiente sintaxis, por ejemplo:

```
--nombre-variable: valor;
```

Por ejemplo:

```
--penguin-skin: grey;
```

Podemos usar esta variable y asignar su valor a otras propiedades CSS de la siguiente manera:

```
background: var(--penguin-skin);
```

Podemos añadir un valor de sustitución (fall-back) de la siguiente manera:

```
background: var(--penguin-skin, pink);
```

Al crear una variable CSS, ésta está disponible para usar tan solo dentro del selector en el que se ha creado, así como para cualquiera de sus descendientes.

```
:root {
    --penguin-skin: grey;
}
```

Para poder hacer uso de la herencia, las variables CSS son definidas normalmente en el elemento :root. :root es un selector de pseudoclase que selecciona el elemento raíz del documento (normalmente el elemento html). Al crear nuestras variables en :root, estas estarán disponibles globalmente. (:root no afecta a la especifidad normal).

Linear gradient function

Crea una imagen que consiste en una transición progresiva entre dos o más colores a lo largo de una línea recta.

```
linear-gradient(gradient direction, color1, color2,...)
```

Gradient direction debe ir en grados (deg) o alguna unidad similar como vuelta (turn), o también se puede indicar la dirección: to top (0deg), to bottom (180deg), lo left (270deg) o to right (90deg).

Ejemplo:

```
background: linear-gradient(0.25turn, #3f87a6, #ebf8e1, #f69d3c);
```

Repeating linear gradient

Similar al anterior, pero hay que añadir los puntos de comienzo y parada de cada color. Por ejemplo:

```
 background: repeating-linear-gradient(red 0px, red 20px, blue 20px, blue
25px);
```

Esta función es un poco más compleja de usar que el ejemplo anterior y tiene otros modos de uso. Más info: https://developer.mozilla.org/en-US/docs/Web/CSS/gradient/repeating-linear-gradient

@keyframes y propiedades de animación

Las propiedades de animación controlan cómo la animación debería comportarse y la regla @keyframes controla qué pasa durante la animación. Hay 8 propiedades de animación en total. Las más importantes:

Animation-name

Establece el nombre de la animación, el cual es luego usado por @keyframes para decirle a CSS qué regla va con qué animación.

Animation-duration

Establece la duración temporal de la animación.

Ejemplo de uso:

```
#anim {
    animation-name: colorful;
    animation-duration: 3s;
}

@keyframes colorful {
    0% {
        background-color: blue;
    }

    100% {
        background-color: yellow;
    }
```

```
}
```

Animation-fill-mode

Esta propiedad, cuando es establecida como “forwards”, especifica el estilo aplicado a un elemento cuando la animación ha terminado (100%). Es decir, deja al elemento con este estilo después de haberse cumplido la animación.

```
    animation-fill-mode: forwards;
```

Animation-iteration-count

Nos permite controlar cuantas veces nos gustaría que se repita el loop de la animación. Puede establecerse a: infinite.

```
    animation-iteration-count: infinite;
```

Animation-timing-function

Controla cómo de rápido un elemento animado cambia a lo largo de la duración de la animación. Funciones de aceleración: ease, ease-out, ease-in, linear.

La función cubic bezier

Consiste en cuatro puntos principales de control en un cuadrado 1x1: p0, p1, p2 y p3. P0 (0,0) y p3 (1,1) controlan el punto inicial y final respectivamente. Podemos establecer los valores x e y de los otros dos puntos de la siguiente manera:

```
    animation-timing-function: cubic-bexier(p1x,p1y,p2x,p2y);
```

La coordenada x puede valer entre [0,1], pero la coordenada y puede aceptar valores mayores que 1.

Ejemplo:

```
animation-timing-function: cubic-bezier(0.5, 0.2, 0.7, 1);
```

Tablas

caption-side

Esta propiedad sirve para indicar dónde se pone el título de la tabla. Puede tener los valores: top, bottom, left y right.

empty-cells

Esta propiedad soluciona la carencia del XHTML que, al no dibujar las celdas que estaban vacías, obligaba a ponerun espacio en blanco usando el carácter . Los valores que admite son:

- **show** que permite mostrar los bordes y fondos como en las celdas con contenido.
- **hide** que permite ocultar los bordes y fondos de las celdas vacías.
- **inherit** que permite heredar el valor de empty-cells que tenga su elemento padre.

border-collapse

Permite establecer el modo en el que se dibujan los bordes de las tablas: **separate** (separados), **collapse** (juntos) e **inherit**. En el modo separate, cada celda está rodeada por su borde haciendo el efecto de un borde con una línea doble, mientras que, en el modo collapse las celdas contiguas comparten sus bordes.

border-spacing

Permite establecer la separación entre celdas contiguas. Para hacerlos se indica el valor del espaciado horizontal seguido del valor del espaciado vertical. Si se escribe un único valor, la separación horizontal y vertical serán iguales.

table-layout

Permite definir el modo en el que el navegador dibujará la tabla ya que puede hacerse de dos formas. Los valores que admite son:

- **fixed:** dibuja la tabla basándose en las medidas establecidas en el código fuente. Con este valor se consigue que el sistema trabaje más rápido.
- **auto:** dibuja las tablas basándose en el contenido de sus celdas. Es el valor por defecto.

Listas

list-style-type

Permite elegir el marcador visual de la lista asignando a la propiedad uno de los siguientes valores: **none** (eliminar el marcador), **square** (cuadrado), **disc** (círculo), **circle** (circunferencia), **lower-roman** (números romanos en minuscula), **lower-alpha** (letras en minúscula), **upper-alpha** (letras en mayúscula), etcétera.

list-style-image

Permite especificar una imagen como marcador. Para ello deberemos indicar la dirección o URL donde se encuentra la imagen. Cuando se usa esta propiedad conviene declarar también la propiedad **list-style-type** en prevención de un fallo en la localización de la imagen. Esto lo podríamos realizar, también empleando la propiedad background del elemento li.

list-style-position

Establece la posición del marcador de los elementos de la lista. Se puede colocar el marcador dentro del área de contenido con lo que todas las líneas de este elemento estarán alineadas por la izquierda (incluida la que lleva el marcador) o, se puede colocar fuera del área de contenido (como en una sangría francesa). Los valores que permiten posicionar el marcador son: **inside** (dentro) y **outside** (fuera).

list-style

Al igual que ocurría con otras propiedades que se vieron anteriormente, esta propiedad permite configurar las listas estableciendo, de forma abreviada y en cualquier orden, el valor de una o más de las propiedades individuales vistas en este apartado.

Imágenes

object-fit

Esta propiedad se utiliza para especificar cómo una imagen <img> o un video <video> debe redimensionarse para ocupar su contenedor.

fill

Valor por defecto. La imagen se redimensiona para ocupar la dimensión del elemento. Esto hará que la imagen se distorsione si las proporciones no coinciden.

contain

La imagen mantendrá su relación de aspecto, pero se redimensiona para ocupar el tamaño del elemento.

cover

La imagen mantiene su relación de aspecto y rellena la dimensión del elemento. La imagen será recortada para caber dentro.

none

La imagen no se redimensiona.

scale-down

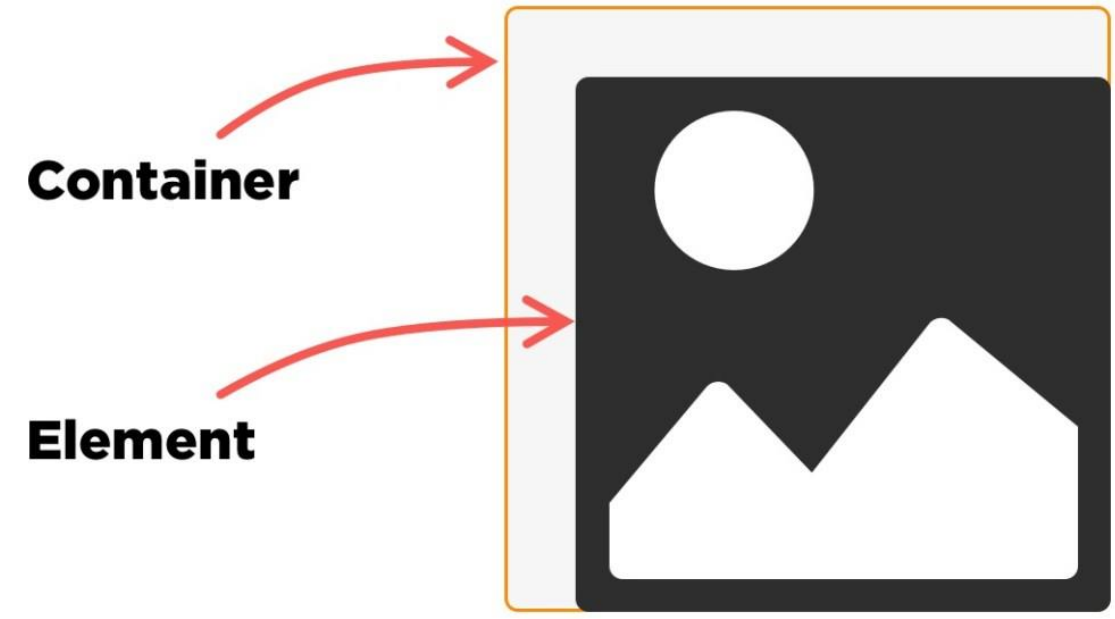
La imagen se escala a la versión más pequeña de **none** o **contain**.

object-position

Esta propiedad especifica como una imagen o video debe posicionarse dentro de su contenedor.

Ejemplo:

```
img {
  object-position: right bottom;
}
```



Sintaxis:

```
/* Positional values */
object-position: 50% 50%; /* default position */
object-position: right bottom;
object-position: 20px 95px;
object-position: center 20px; /* mix and match */
object-position: 60% top; /* mix and match */

/* Global values */
object-position: inherit;
object-position: initial;
object-position: unset;
```

Centrado de un elemento dentro de otro

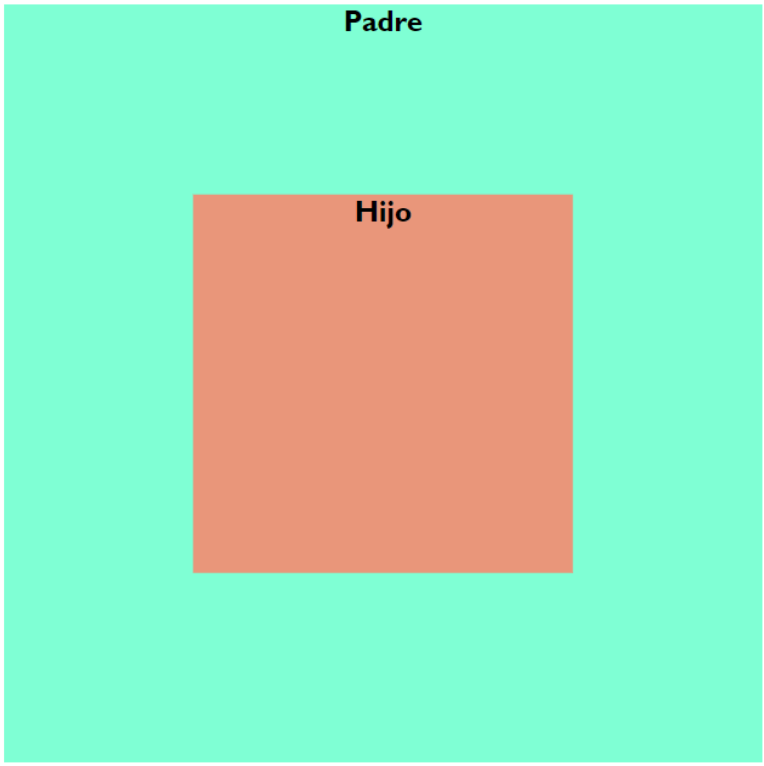
Código HTML:

```
    <section>
        <div id="padre">Padre
            <div id="hijo">Hijo</div>
        </div>
    </section>
```

Código CSS:

```
        #padre {
            background-color: aquamarine;
            /*El elemento  padre debe tener un ancho y alto definido*/
            width: 500px;
            height: 500px;
            /*Establecemos su propiedad position como relative*/
            position: relative;
        }

        #hijo {
            background-color: darksalmon;
            width: 250px;
            height: 250px;
            /*En el hijo, establecemos la propiedad position como
absolute*/
            position: absolute;
            /*Con las propiedades top, left y transform (si el hijo tiene
ancho y alto definido):*/
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
        }
```

Resultado:



Explicación:

1. **top: 50%; y left: 50%;** colocan el punto de referencia del elemento hijo en el centro del padre, es decir, su esquina superior izquierda está en el centro del padre.
2. **transform: translate(-50%, -50%);** desplaza el elemento hijo hacia arriba y hacia la izquierda en un 50% de su propio ancho y alto, lo que efectivamente lo coloca centrado dentro del padre.

Alternativamente:

```
        #hijo {
            background-color: darksalmon;
            /*En el hijo, establecemos la propiedad position como
absolute*/
            position: absolute;
            /*Con las propiedades top left y margin (si el hijo no tiene
el alto y ancho definido):*/
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            margin: auto
        }
```

Explicación:

1. **top: 0;, left: 0;, right: 0;, bottom: 0;** establecen que los bordes superior, izquierdo, derecho y inferior del elemento hijo estén pegados a los bordes superior, izquierdo, derecho y inferior del padre, respectivamente.
2. **margin: auto;** distribuye automáticamente el espacio sobrante en los ejes horizontal y vertical, lo que efectivamente centra el elemento hijo dentro del padre.

CSS Responsive y Flexbox

Flexbox

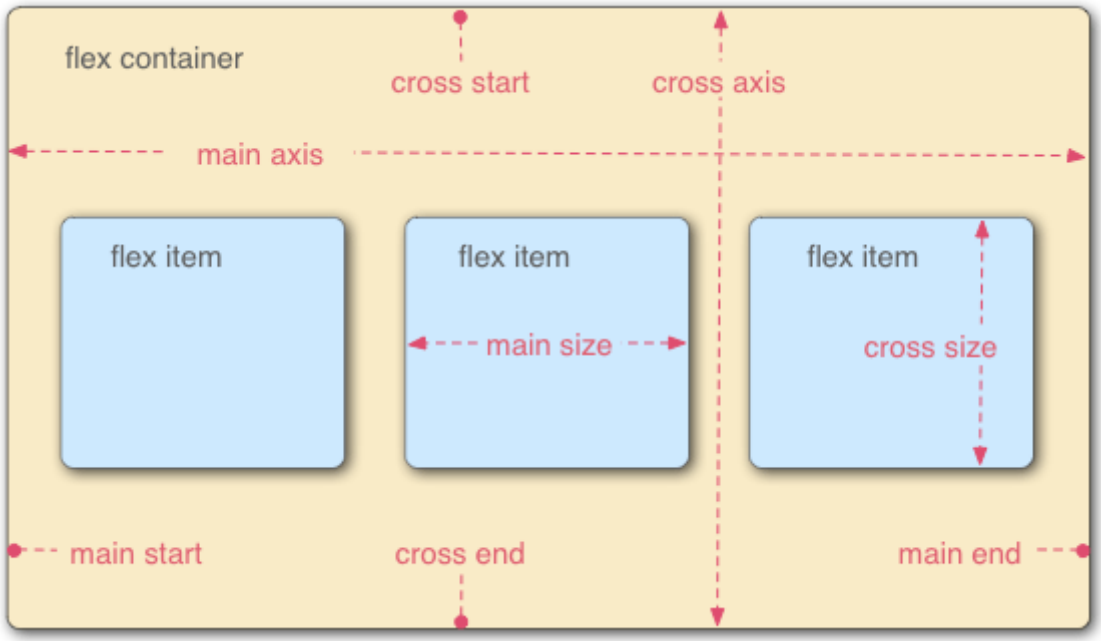
Flexbox es un método de diseño de página unidimensional para compaginar elementos en filas o columnas. Los elementos de contenido se ensanchan para rellenar el espacio adicional y se encogen para caber en espacios más pequeños.

Flexbox nos permite distribuir el espacio de manera dinámica sobre elementos de tamaño desconocido, de ahí el término "flex".

Para poder utilizarlo lo primero que hay que hacer es habilitarlo mediante la propiedad "display" de la siguiente manera en el elemento contenedor:

```
    display: flex;
```

El modelo flex



Flex-direction

Por defecto, el "main axis" o eje principal, va de izquierda a derecha y el "cross axis" o eje transversal va de arriba a abajo. La primera propiedad en ver es flex-direction, que nos permite justamente establecer cuál es el eje principal (horizontal o vertical)

```
    flex-direction: row;
```

"row" es el valor por defecto, establece el eje principal de izquierda a derecha.

Eje principal de derecha a izquierda:

```
    flex-direction: row-reverse;
```

Que el eje principal sea el “cross axis” (arriba - abajo):

```
    flex-direction: column;
```

Y de abajo hacia arriba:

```
    flex-direction: column-reverse;
```

Justify-content

Distribuye la posición del contenido a lo largo del eje principal. La opción por defecto es:

```
    justify-content: flex-start;
```

Si nuestro main axis es por defecto, “flex-start” es alinear el contenido la parte izquierda. Por lo tanto, flex-end será alinear el contenido a la parte derecha. Pero hay que tener en cuenta que si flex-direction fuera row-reverse, la alineación sería justo al revés.

```
    justify-content: flex-end;
```

```
    justify-content: center;
```

Alineará el contenido en el centro.

```
    justify-content: space-between;
```

Coge todo el espacio extra y lo distribuye entre los elementos (pero no en los extremos).

```
justify-content: space-around;
```

Es como el anterior pero esta vez el espacio se reparte también en los extremos, de manera que les asigna la mitad del espacio entre los elementos a cada extremo:

```
justify-content: space-evenly;
```

Como los anteriores pero esta vez se reparte todo el espacio sobrante de manera equitativa tanto entre los elementos como en los extremos:

Recuerda que todo siempre es relativo al “main axis”, que podría no ser horizontal sino vertical.

Flex-wrap

Determinará si nuestros elementos van a ser “envueltos” a lo largo del eje principal y saltar a una nueva línea (o a una nueva columna si nuestra alineación es vertical). Con esto garantizamos que los elementos conserven su tamaño original y no sean “escachados” para encajar en el elemento contenedor.

```
section {
    height: 200px;
    background-color: grey;
    display: flex;
    flex-direction: column;
    flex-wrap: wrap;
}
```


```
    flex-wrap: wrap-reverse;
```

La alineación sigue siendo la misma (column, de arriba hacia abajo a lo largo del eje transversal), pero los elementos se alinean hacia el lado contrario (en este caso la derecha).

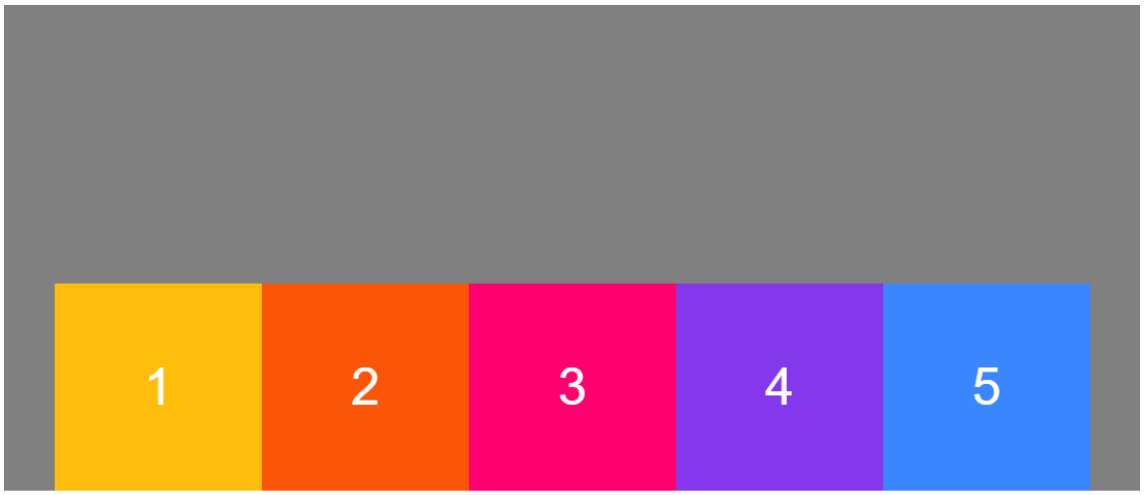
Align-items

Distribuye los elementos a lo largo del eje transversal.

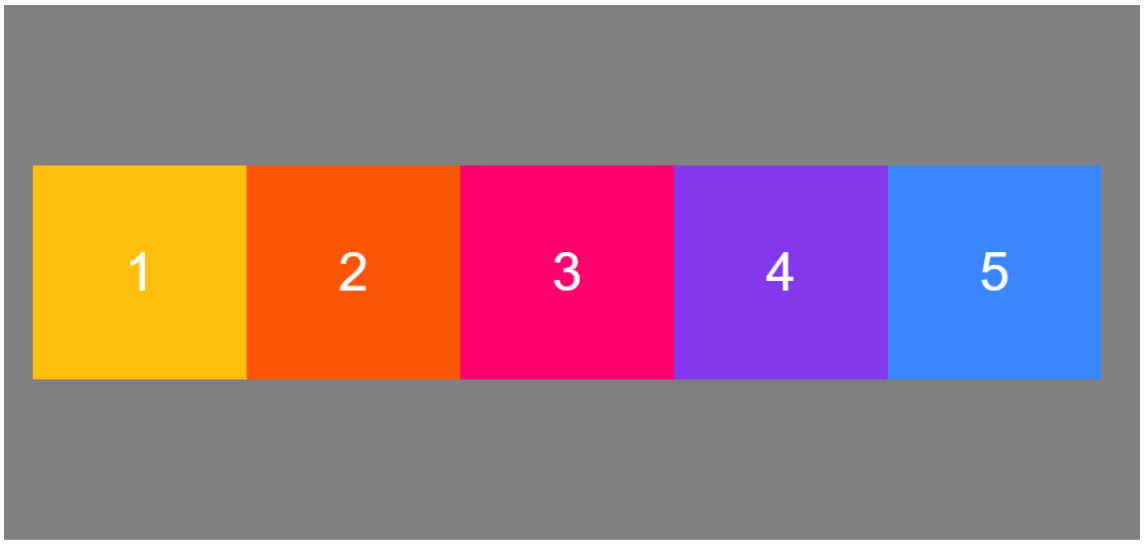
```
    section {

    height: 500px;
    background-color: grey;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: flex-start;
}
```

Alinea los elementos al principio del eje transversal.


```
    align-items: flex-end;
```



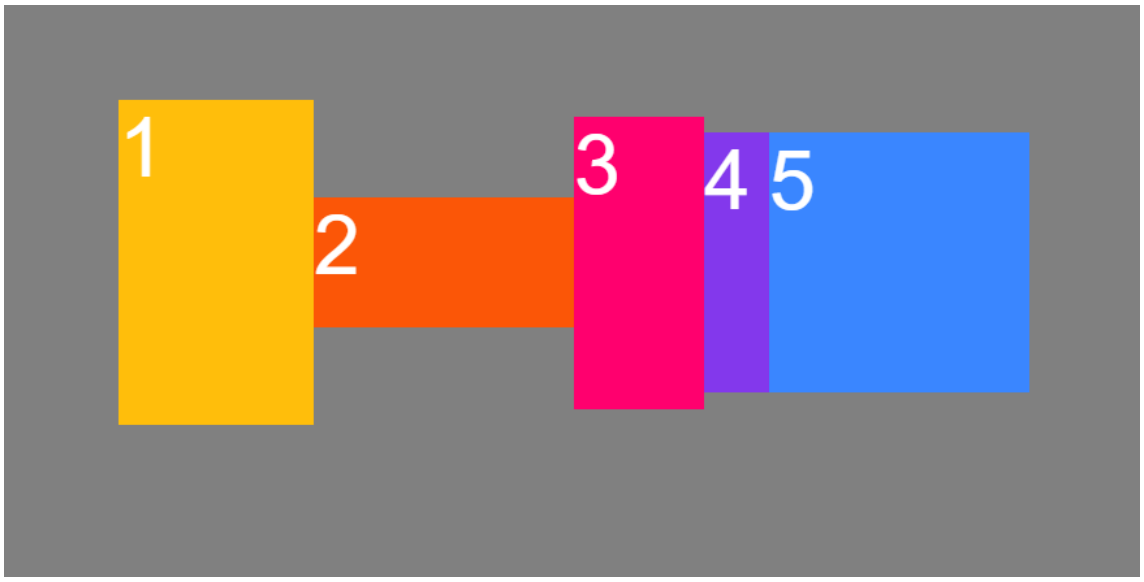

```
    align-items: center;
```




```
    align-items: center;
```

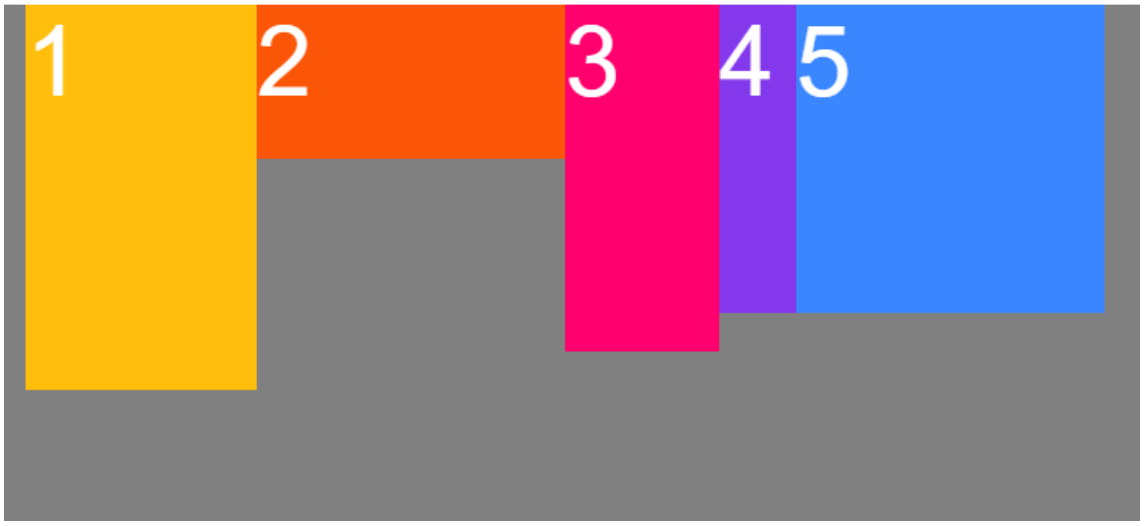
Además, suponiendo este caso:

```
section {
    height: 500px;
    background-color: grey;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;

}
```



Podemos usar:

```
    align-items: baseline;
```



Lo que hace es alinear los elementos acordes al “baseline” del texto.

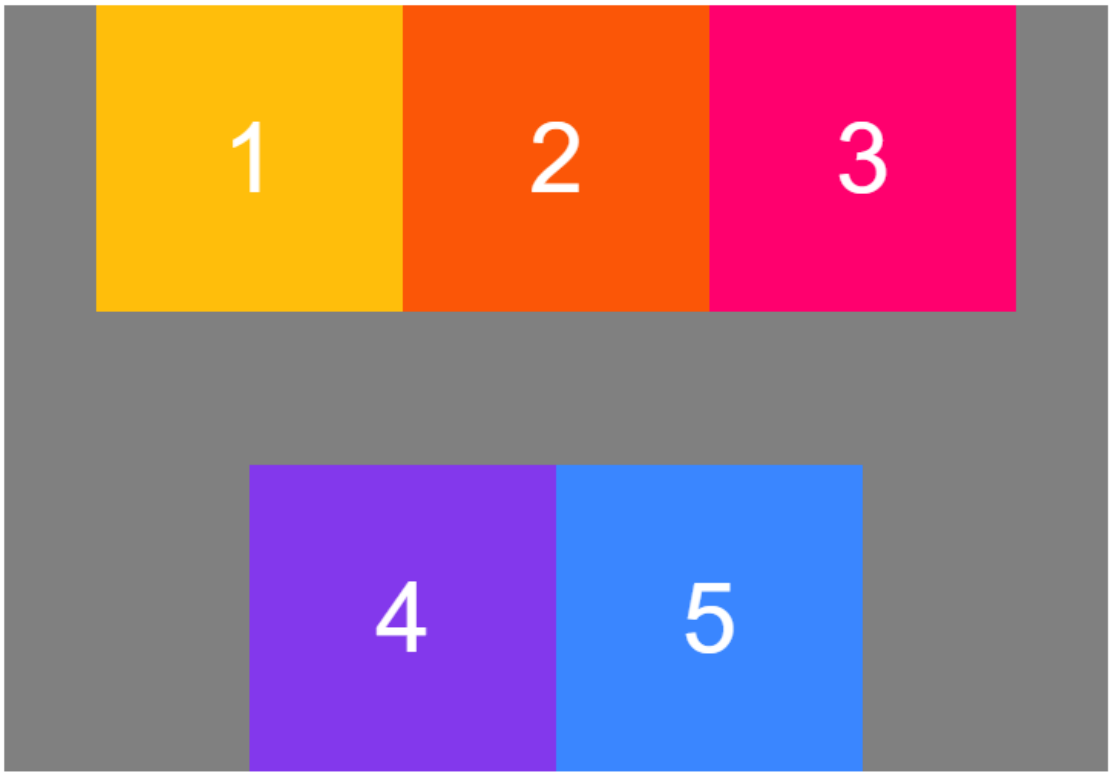
Align-content & Align-self

Align-content

Align-content se usa para justificar el contenido acorde al eje transversal, pero solo funciona y tiene sentido cuando tenemos activado flex-wrap:

```
section {
    width: 720px;
    height: 500px;
    background-color: grey;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    flex-wrap: wrap;
```

```
    align-content: space-between;
}
```



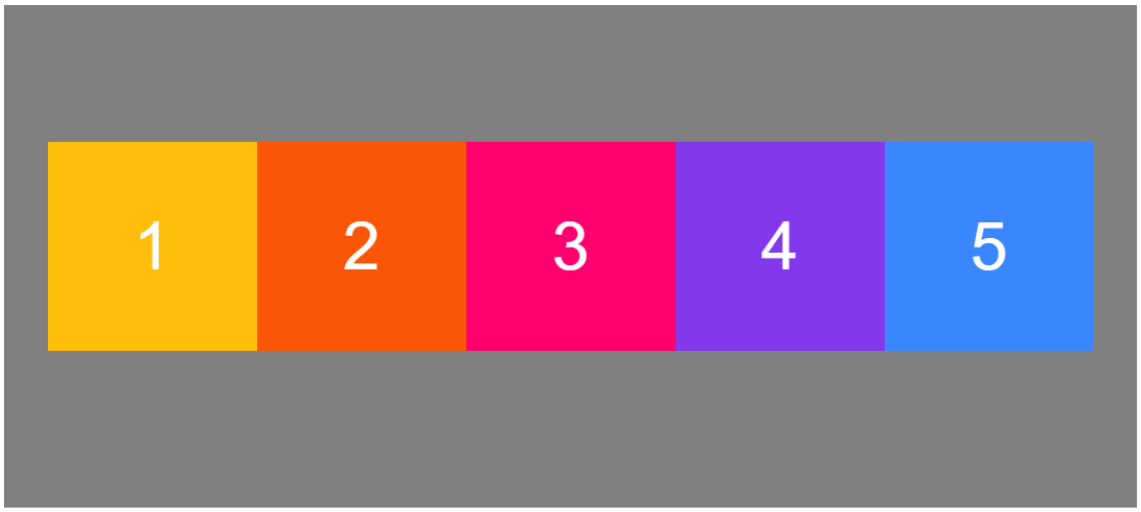
Además, contamos con las alineaciones habituales: space-around, space-evenly, flex-start, flex-end, center, etc.

Aling-self

Aling-self, es muy similar a align-items, excepto que esta vez aplicamos la propiedad sobre un único ítem o elemento. Por lo tanto, hay que aplicar la propiedad, no sobre el elemento contenedor, como hasta ahora, sino al elemento en sí. Por ejemplo:

Elemento contenedor

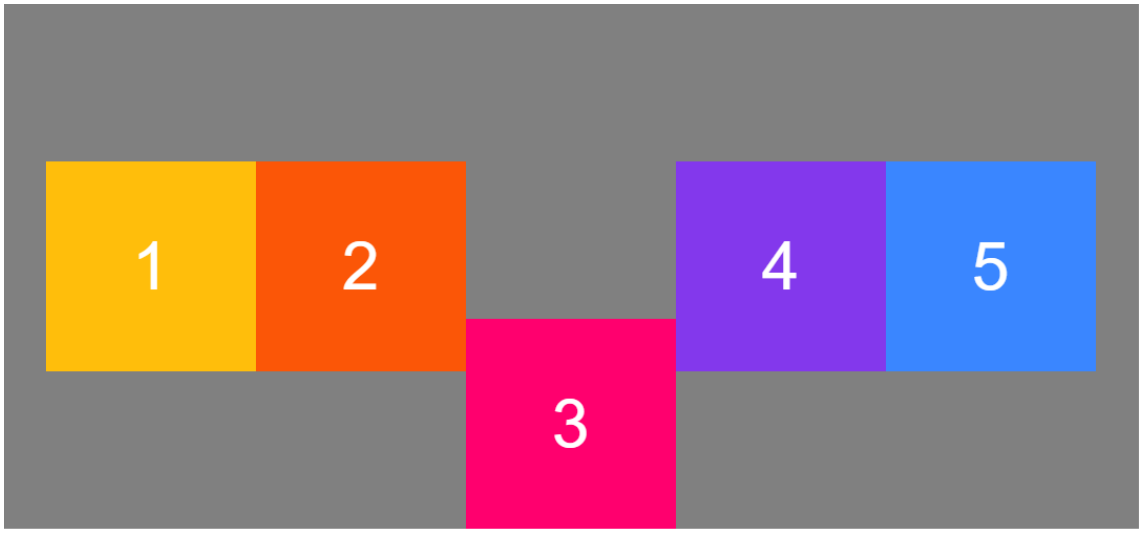
```
section {
    height: 500px;
    background-color: grey;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
}
```



Y ahora vamos a aplicar la propiedad:

```
div:nth-of-type(3) {
    background-color: #FF006E;
    align-self: flex-end;
}
```

a la caja número 3:



Lo mismo con el resto de propiedades: flex-start y center.

Flex-basis, Grow & Shrink

Flex-basis

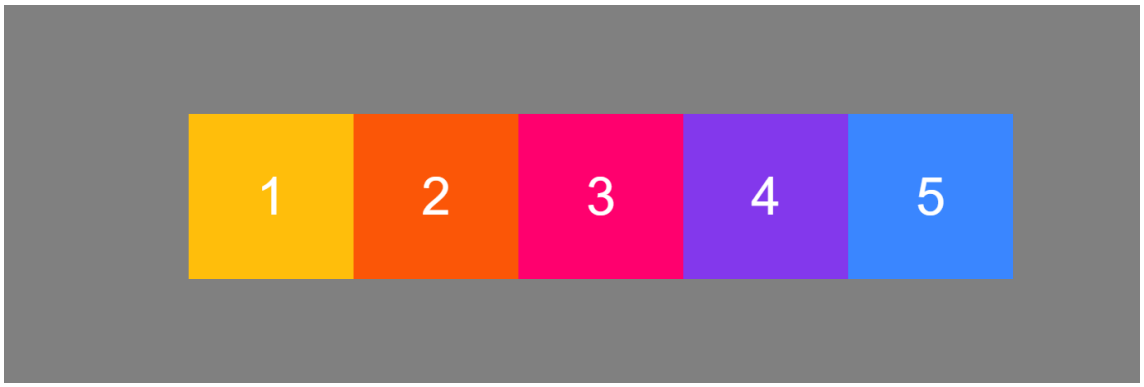
Flex-basis define el tamaño inicial de un elemento antes de que se distribuya espacio adicional. Funciona a lo largo del eje principal.

```
    flex-basis: 400px;
```

Flex-grow

Flex-grow controla la cantidad de espacio disponible que un elemento debería cogerse. Acepta solo valores no unitarios.

Por ejemplo, partiendo de:




```
div {
    width: 200px;
    height: 200px;
    font-size: 4em;
    font-family: Arial, Helvetica, sans-serif;
    color: white;
    display: flex;
    align-items: center;
    justify-content: center;
    flex-basis: 200px;
}

section {
    height: 500px;
    background-color: grey;
    display: flex;
    flex-direction: row;
    justify-content: center;
    align-items: center;
    flex-wrap: wrap;
}
```

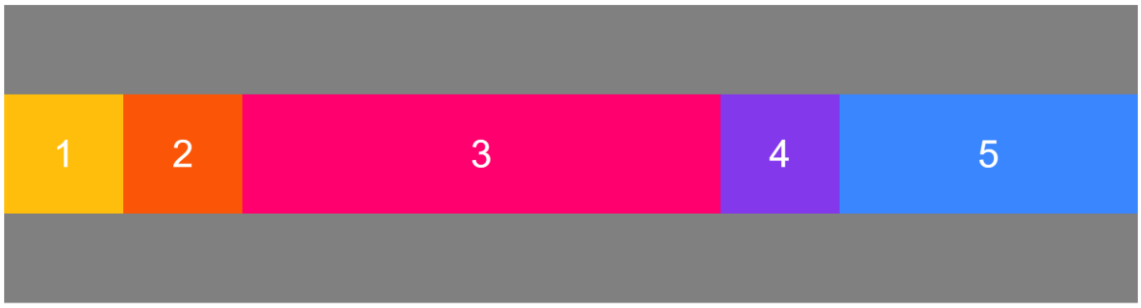
Le vamos a dar al primer cuadrado un valor de 1 en flex-grow.

```
div:nth-of-type(1) {
    background-color: #FFBE0B;
    flex-grow: 1;
}
```

Podemos ver como lo que ha hecho es cogerse todo el espacio adicional para sí mismo.

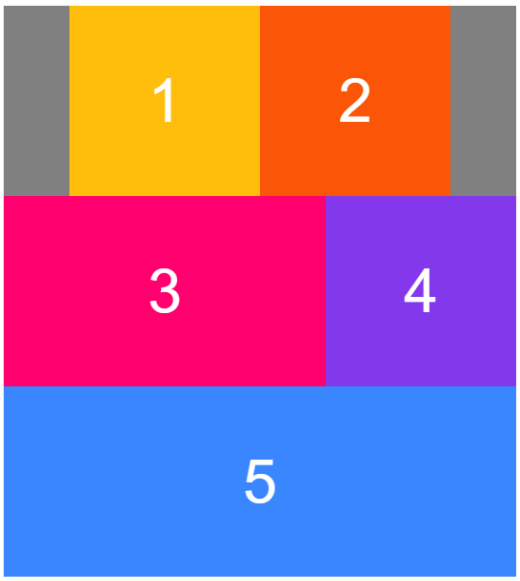
Si, por ejemplo, hacemos lo mismo para todos los cuadrados, los cuadrados rellenarán todo el espacio disponible (en el main axis) de manera equitativa. Así que, efectivamente, el valor que le estamos dando a la propiedad flex-grow no es sino una proporción.

Otro ejemplo, le damos valor 2 al tercer cuadrado y 1 al quinto.



Vemos que se han distribuido el espacio en la proporción 2 a 1.

Notar que como le hemos activado al contenedor “flex-wrap” si vamos haciendo pequeña la ventana del navegador:



Flex-shrink

Flex-shrink, cuando los elementos son mayores que el elemento contenedor, empequeñecen acorde a esta propiedad. Funciona como una especie de flex-grow a la inversa. Si se le pone valor 0 el elemento tendrá su tamaño predeterminado.

Flex

La propiedad **flex** funciona como un atajo para todas las propiedades anteriores (flex-basis, flex-grow y flex-shrink).

Sintaxis y ejemplos

```
/* One value, unitless number: flex-grow
flex-basis is then equal to 0. */
flex: 2;

/* One value, width/height: flex-basis */
flex: 10em;
flex: 30%;
flex: min-content;

/* Two values: flex-grow | flex-basis */
flex: 1 30px;

/* Two values: flex-grow | flex-shrink */
flex: 2 2;

/* Three values: flex-grow | flex-shrink | flex-basis */
flex: 2 2 10%;
```

Diseño responsive y Media Queries

El diseño responsive es un formato de programación que permite ajustar un sitio web automáticamente al tamaño y disposición de los dispositivos de sus usuarios. Los sitios web responsive cambian para ofrecer la mejor experiencia a los visitantes desde sus teléfonos inteligentes, tabletas o computadoras de escritorio.

Una de las maneras en las que esto se puede hacer posible es mediante el uso de media queries. Estas nos permiten modificar nuestros estilos dependiendo de algún parámetro en particular, como por ejemplo el ancho de la pantalla o el tipo de dispositivo.

Media queries

La regla @media asocia un grupo de declaraciones anidadas, en un bloque CSS delimitado por llaves, con una condición definida por un media query. La regla-at @media puede ser usada no solo en el nivel superior de la hoja de estilos, sino también dentro de cualquier grupo de reglas condicionales.

Sintaxis:

```
@media (condiciones) {
   selectores y propiedades;
}
```

Ejemplo:

```
@media (width:360px) {
   div {
       color:red;
   }
```

```
}
```

Las propiedades más comunes que se usan en media queries son: width, height y orientation. La regla anterior nos está diciendo que cuando el ancho sea de exactamente 360px todos los elementos div serán de color rojo.

Más útil y común que usar solo width, es usar **min-width** o **max-width.** Con min-width la propiedad se "activará" desde ese valor para arriba (width mayor o igual al valor indicado), con max-width es justo, al contrario.

Hay que aclarar que aquí el ancho mínimo o máximo hace referencia al ancho del **viewport.** Un viewport representa la región poligonal (normalmente rectangular) en gráficas de computación que está siendo visualizada en ese instante. En términos de navegadores web, se refiere a la parte del documento actualmente visible en la ventana (o la pantalla, si el documento está siendo visto en modo pantalla completa). El contenido fuera del viewport no es visible en la pantalla hasta que sea desplazado dentro de él.

```
@media (min-width:1000px) {
    section {
        background-color: red;
    }
}
```

También se pueden concatenar instrucciones con "and":

```
@media (min-width:900px) and (max-width: 1000px) {
    section {
        background-color: red;
    }
}
```

Enlace útil: https://devfacts.com/media-queries-breakpoints-2023/

Trucos: Hacer que una imagen sea responsive y otros trucos

Hacer una imagen responsive

Hacer que una imagen sea responsive es muy fácil utilizando CSS. Si siempre añadimos:

```
img {
    max-width: 100%;
    height: auto;
}
```

Establecer max-width al 100% hará que la imagen nunca sea más ancha que su contenedor. Establecer la altura como "auto", hará que la imagen mantenga su proporción original y no se deforme.

Retina image para grandes resoluciones

La manera más simple de hacer que las imágenes se visualicen bien en dispositivos de gran resolución (como retina display de los Macbook Pro) es definir sus anchuras y alturas como la mitad de los del archivo original.

Hacer la tipografía responsive

El truco está en trabajar con unidades de viewport (vh y vw) en lugar de em o rem.

CSS Grid Layout

CSS Grid layout contiene funciones de diseño dirigidas a los desarrolladores de aplicaciones web. El CSS grid se puede utilizar para lograr muchos diseños diferentes. También se destaca por permitir dividir una página en áreas o regiones principales, por definir la relación en términos de tamaño, posición y capas entre partes de un control construido a partir de primitivas HTML.

Al igual que las tablas, el grid layout, permite a un autor alinear elementos en columnas y filas. Sin embargo, con CSS grid son posibles muchos más diseños y de forma más sencilla que con las tablas. Por ejemplo, los elementos secundarios de un contenedor de cuadrícula podrían posicionarse para que se solapen y se superpongan, de forma similar a los elementos posicionados en CSS.

Resumiendo: CSS GRID nos ayuda a construir *layouts* complejos que sean *responsive*.

"Activar" el grid (contenedores)

Para hacer que cualquier elemento HTML contenedor en un grid.

```
display: grid;
```

Añadiendo estructura al grid (grid-template-columns)

La propiedad **grid-template-columns** se usa para añadir columnas al grid. El número de parámetros indica el número de columnas en el grid, y el valor de cada uno, la anchura de cada columna.

Grid-template-rows

Como el anterior, pero para las filas.

Unidades

Se pueden usar tanto las unidades relativas como las absolutas, y, además;

- **fr:** establece las columnas o las filas a una fracción del espacio disponible.
- **auto:** establece las columnas o las filas al ancho o la alturade su contenido automáticamente.
- **%:** ajusta la columna o la fila al porcentaje del ancho de su contenedor.

Grid-column-gap

Añade espacio (hueco) entre las columnas. Se pueden emplear las unidades habituales.

Grid-row-gap

Añade espacio entre las filas. Se pueden emplear las unidades habituales.

*Shorthand: grid-gap

Si se pone un único valor establece ese valor a grid-column-gap y grid-row-gap, si se ponen dos valores, el primero es para las filas y el segundo para las columnas.

Grid-column (ítems)

Se usa en los elementos dentro del grid contenedor. Con esta propiedad establecemos el espacio que queremos que ocupen dentro del grid.

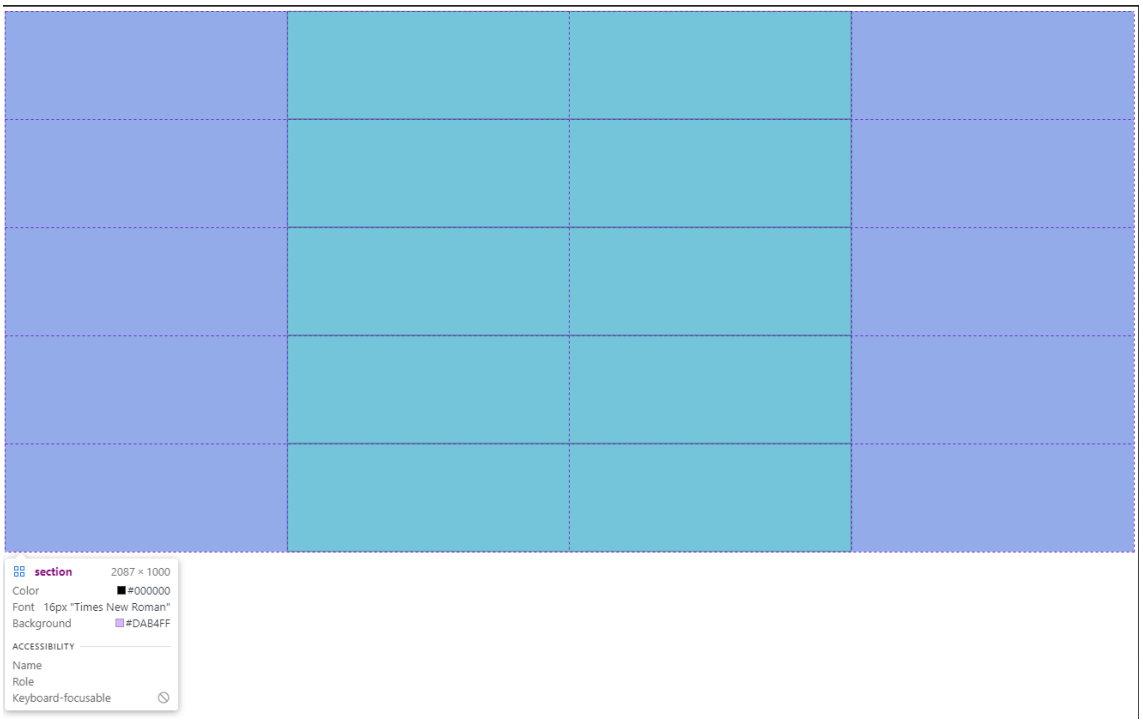
Sintaxis: **grid-column: main start / main end;**

Ejemplo:

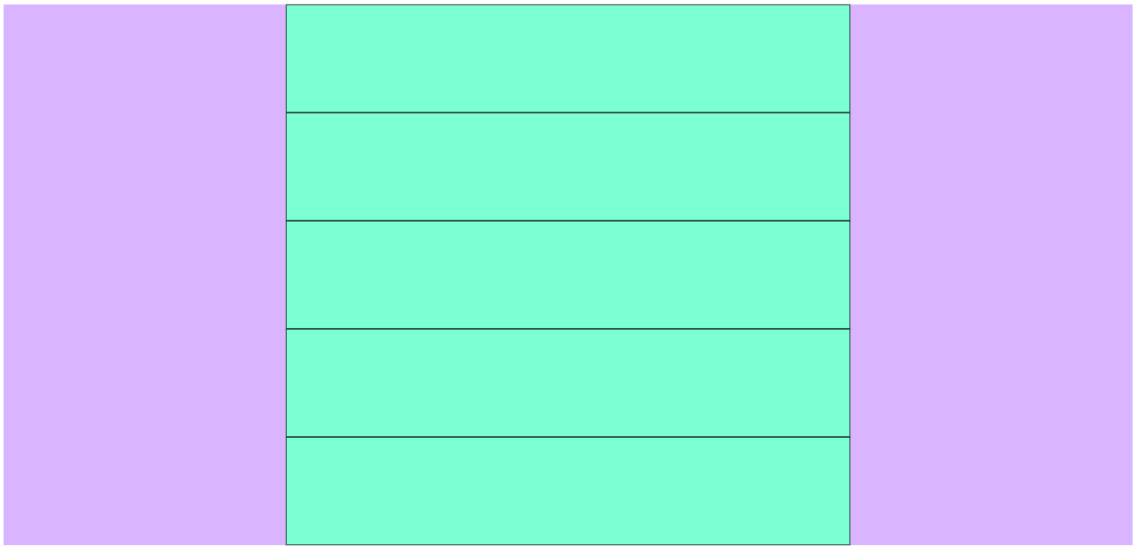
```
div {
    width: 100%;
    height: 200px;
    background-color: rgb(125, 255, 212);
    box-sizing: border-box;
    border: 1px black solid;
    grid-column: 2 / 4;
}

section {
    width: 100%;
    display: grid;
    grid-template-columns: 25% 25% 25% 25%;
    /*grid-column-gap: 5%;*/
    background-color: rgb(218, 180, 255);
}
```

Tenemos un elemento section con 5 elementos div en su interior. Hemos establecido un grid dentro de section de 4 columnas donde cada una ocupará el 25% del espacio (4 columnas iguales que se cogen todo el espacio disponible). Los elementos div (aguamarina) ocupan el 100% de la anchura del espacio disponible y tienen una altura de 200px, para poder diferenciarlos se le ha añadido un borde de 1 px.

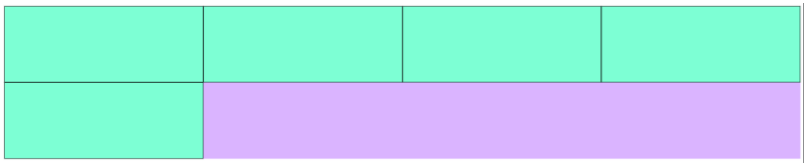
En la imagen siguiente podemos ver la distribución del grid creado, cada cuadrícula es considerada una celda "**cell**" en inglés.

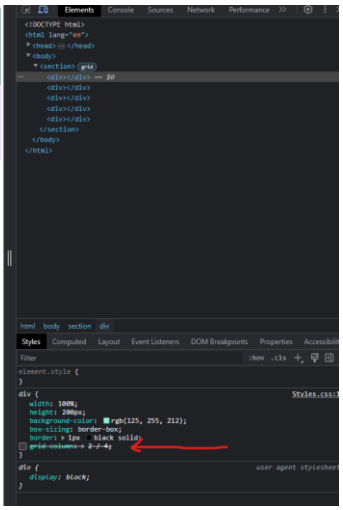


Como se aprecia se ha indicado un grid-colum: 2/4, es decir, que los div van a ocupar desde el inicio de la columna dos al inicio de la 4.



Si desactivamos esa opción vemos como se distribuyen a lo largo de todo el grid:





Grid-row

Funciona como grid-column pero para las filas.

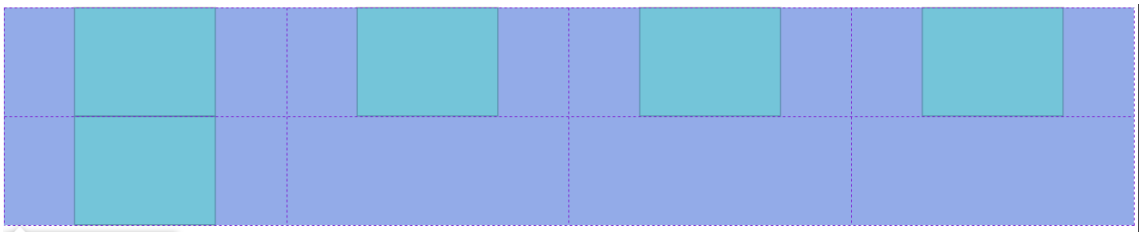
Justify-self

Alinea la posición del contenido dentro de su celda horizontalmente (su valor es “stretch” por defecto). Valores:

- Start (izquierda)
- Center
- End

Vamos a tomar el ejemplo anterior, pero establecer que el ancho de los div sea del 50% para poder visualizar bien esta propiedad. Además, le añadimos la propiedad:

```
    justify-self: center;
```



Podemos ver como el contenido se alinea en el centro de la celda.

Align-self

Como el anterior pero el contenido se alinea verticalmente.

Justify -items (contenedores)

Alinea todos los ítems horizontalmente al mismo tiempo.

Align-items

Como el anterior pero verticalmente.

Grid-template-areas

Agrupa celdas del grid juntas en un área y le da al área un nombre personalizado. Cada palabra representa una celda y cada par de comillas << “ >> representa una fila. Por ejemplo:

```
    grid-template-areas:
        "header header header"
        "advert content content"
        "advert footer footer";
```

Podemos colocar nuestros ítems en nuestra área personalizada referenciándola con el nombre que le hemos dado usando:

Grid-area (ítems)

Por ejemplo, en el siguiente código:

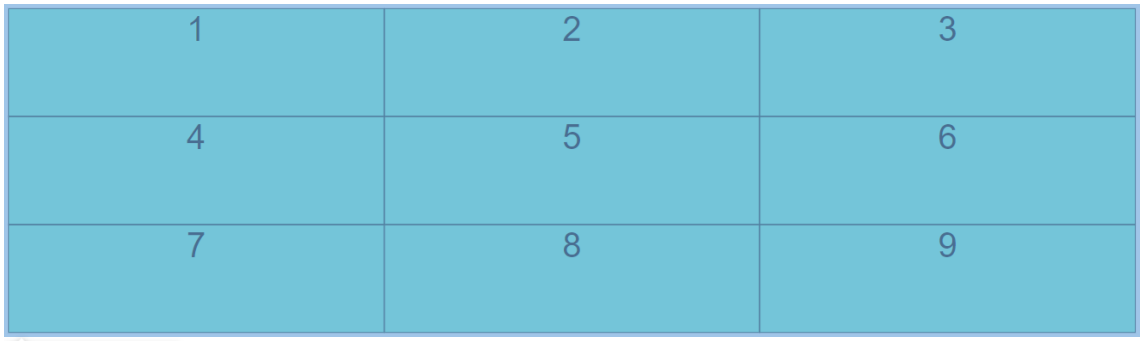
HTML

```
<body>
    <section>
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
        <div>5</div>
        <div>6</div>
        <div>7</div>
        <div>8</div>
        <div>9</div>
    </section>
</body>
```

CSS

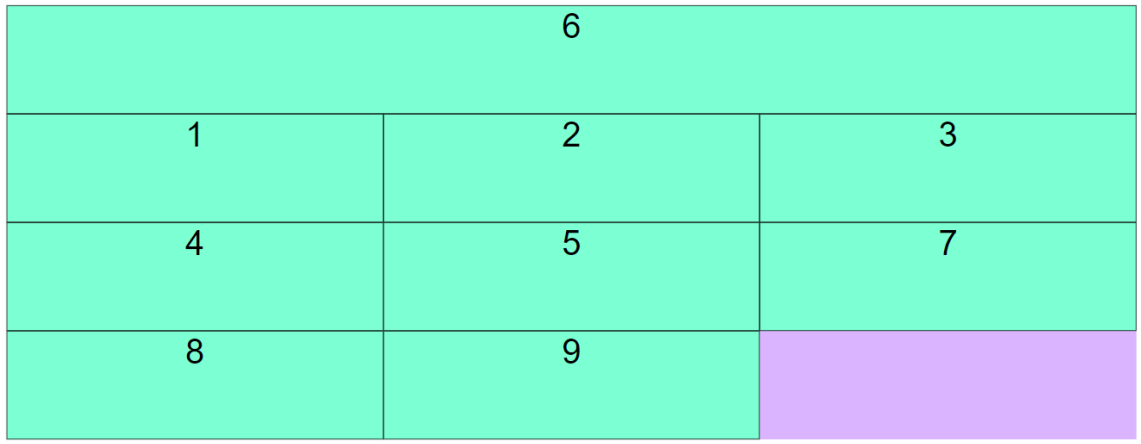
```
div {
    width: 100%;
    height: 200px;
    background-color: rgb(125, 255, 212);
    box-sizing: border-box;
    border: 1px black solid;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 4em;
    text-align: center;
    /*grid-column: 2 / 4;*/
    /*justify-self: center;*/
}

section {
    width: 100%;
    display: grid;
    /*grid-template-columns: 25% 25% 25% 25%;
    grid-column-gap: 5%;
    justify-items: center;*/
```

```
    background-color: rgb(218, 180, 255);
    grid-template-areas:
        "header header header"
        "advert content content"
        "advert footer footer";
}
```



Si colocamos el div número 6 en la sección “header”

```
div:nth-of-type(6) {
    grid-area: header;
}
```



Si el grid no tiene un “area-template” al que referirse, podemos crear uno sobre la marcha para un ítem:

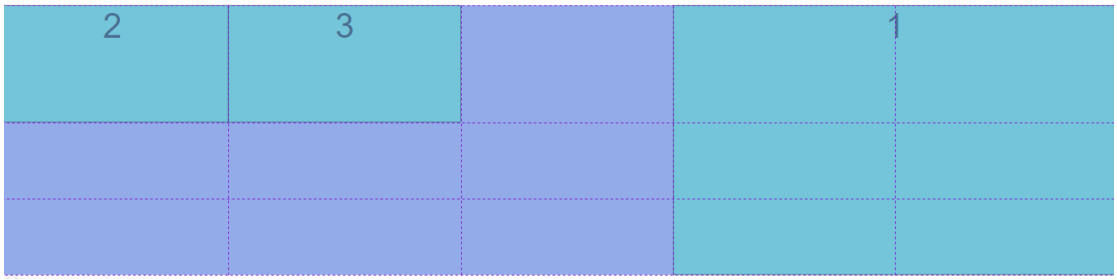
```
item {
    grid-area: 1/2/3/4;
}
```

Donde:

- 1: Línea donde empieza la fila
- 2: Línea donde empieza la columna
- 3: Línea donde termina la fila
- 4: Línea donde termina la columna

Ejemplo:

```
div {
    width: 100%;
    background-color: rgb(125, 255, 212);
    box-sizing: border-box;
    border: 1px black solid;
    font-family: Arial, Helvetica, sans-serif;
    font-size: 4em;
    text-align: center;
    /*grid-column: 2 / 4;*/
    /*justify-self: center;*/
}

section {
    width: 100%;
    height: 500px;
    display: grid;
    /*grid-template-columns: 25% 25% 25% 25%;
    grid-column-gap: 5%;
    justify-items: center;*/
    background-color: rgb(218, 180, 255);
}

div:nth-of-type(1) {
    grid-area: 1/4/4/6;
}
```



Enlace de interés: https://css-tricks.com/snippets/css/complete-guide-grid/#aa-introduction

The repeat function

Sirve para especificar el número de veces que queremos que nuestra columna o fila se repita. Por ejemplo:

```
    grid-template-rows: repeat(5, 50px);
```

Crea 5 filas de 50px de una sentada.

También podemos usar la función repeat() y añadir otros valores al mismo tiempo, por ejemplo:

```
    grid-template-columns: repeat(2, 1fr 100px) 200px;
```

Es equivalente a haber puesto:

```
    grid-template-columns: 1fr 100px 1fr 100px 200px;
```

Minmax function

Limita el tamaño de los ítems cuando el contenedor grid cambia de tamaño. Para hacerlo necesitamos especificar el tamaño del rango aceptable para nuestro ítem. Por ejemplo:

```
    grid-template-columns: 100px minmax(100px, 250px);
```

Crea dos columnas. Una de 100px y la segunda con un mínimo de 100px y un máximo de 250px.

Auto-fill

Permite insertar automáticamente tantas filas y columnas de un determinado tamaño como sea posible, dependiendo del tamaño del contenedor. Por ejemplo:

```
   repeat(autofill, minmax(60px, 1fr);
```

mientras que auto-fill sigue insertando filas o columnas una vez excedido el tamaño del contenedor, empujando los ítems a un lado,

Auto-fit

Colapsa esas filas o columnas vacías y alarga o encoge los ítems para que quepan en el tamaño del contenedor. Si el contenedor no puede albergar todos los ítems en una fila (o columna), los moverá a la siguiente como hace autofill.

Accesibilidad aplicada (HTML+)

Etiquetas HTML

Audio

La etiqueta <audio></audio> soporta el atributo “control”. Este atributo hace que el navegador muestre botones de play, pause y otros controles por defecto, además de soportar funcionalidades de teclado. Se trata de un atributo booleano, es decir, no necesita ningún valor, solo estar o no presente.

Ejemplo:

```
<audio controls>
      <source src="myAudio.mp3" type="audio/mpeg" />
      <source src="myAudio.ogg" type="audio/ogg" />
</audio>
```

La etiqueta <source> especifica los recursos multimedia para los elementos <picture>, <audio> y <video>.

Accesibilidad de los botones radio

Como los botones tipo radio normalmente vienen en un grupo donde el usuario debe elegir uno, hay una manera de mostrar semánticamente las opciones que forman parte del conjunto: la

etiqueta <fieldset></fieldset> a menudo usa una etiqueta <legend></legend> para proveer de una descripción al grupo.

Date picker

El atributo type=”date”, es un elemento de tipo input hace que se muestre en el campo input cuando este elemento se encuentra “enfocado” (puntero sobre el elemento) en algún navegador.

El atributo acceskey

Este atributo especifica una clave para activar o atraer el foco a un elemento. Al añadirlo podemos hacer la navegación más eficiente a los navegadores que solo usan teclado.

Ejemplo:

```
<button accesskey="b">Enviar</button>
```

El atributo global accesskey provee un indicio para generar un atajo de teclado para el elemento actual. Este atributo consiste en una lista de caracteres separada por espacios (un único punto de código Unicode). El explorador usa el primero que existe en la distribución del teclado de la computadora.

La operación para activar el accesskey depende del explorador y su plataforma: https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/accesskey

El atributo tabindex

El atributo global tabindex indica si su elemento puede ser enfocado, y si participa en la navegación secuencial del teclado (usualmente con la tecla Tab, de ahí el nombre). Acepta un entero como valor, con diferentes resultados que dependen de dicho valor: https://developer.mozilla.org/es/docs/Web/HTML/Global_attributes/tabindex

CSS

Clases útiles que recordar

.sr-only

Cuando queremos esconder contenido que está presente solo para lectores de pantalla, por ejemplo, cuando la información está en un formato visual pero los lectores de pantalla necesitan una alternativa (como una tabla) para poder acceder a los datos.

Este es un ejemplo de reglas CSS que nos ayudan a conseguir esto:

```
.sr-only {
    position: absolute;
    left: -10000px;
    width: 1px;
    height: 1px;
    top: auto;
    overflow: hidden;
}
```

Tan solo necesitamos añadir esta clase al elemento que deseamos ocultar.