APRENDIENDO **CSS**

CON SOYDALTO

El CSS significa Cascading Style Sheets

Primero se crea un archivo **INDEX.HTML** y en la etiqueta de <head> va la etiqueta:

<link rel="stylesheet" type="text/css" href="estilo.css">

Esto significa que el atributo rel que iría "stylesheet" o Hoja de Estilo.

Luego iría el atributo type="text/css" type es que tipee, y lo que va a tipear va a ser texto y css.

Y, por último, iría el atributo href="estilo.css" que href es la ruta del archivo. Nosotros tenemos que crear un archivo.css, por ejemplo: “estilo.css”.

Esto tiene varias formas de poner el **css** en un archivo **HTML**, pero es más correcto es de este modo.

Para que no quede un “Anti-Patrón de Diseño” y que este todos los códigos ordenados.

Estructuras de CSS

selector {

    propiedad:valor;

}

Así es cómo Funciona la estructura de **CSS**,

Con un Selector (sirve para indicar que queremos cambiar de la estructura o código) Que pueden ser:

1. **Universal:** El selector Universal, es decir, selecciona todo lo que está en la página web en él se utiliza el selector: “ \* “

\* {

}

1. **De Tipo:** El selector de tipo, es para indicar que queremos cambiar exactamente, como puede ser un H1 /P /INPUT / etc.

h1 {

}

1. **CLASS:** El selector por clase, es para ponerle propiedad con el atributo class, por ejemplo:

<h2 class="Titulo-h2">El Titulo con Class</h2>

1. **ID:** El selector con ID, es para ponerle atributo en las diferentes etiquetas adentro de los <> Es para identificar un elemento UNICO con su ID o Identificación. Por ejemplo:

<h2 id="element-for">El Título con ID</h2>

1. **Por Atributo:** El selector por Atributo, sirve para identificar una etiqueta por atributos, podemos crear un atributo a un h2 por ejemplo:

<h2 rancio="sii">El Título por Atributo </h2>

1. **Por Descendiente:** El selector Descendiente, sirve para identificar a una etiqueta mediante más etiquetas adentro de otras. Por ejemplo:

<h2><b>El Título por descendiente</b></h2>

1. **Pseudo Clase:** El selector pseudo, sirve para que al pasar por un título o párrafo se ponga de color. Se utilizan hover o active. Por ejemplo:

<h2 class="h2-class"><b>El Título en Pseudo Clase</b></h2>

Con la Propiedad (sirve para indicarle al selector que queremos cambiar como el tamaño, color, lugar, etc.)

Que pueden ser:

1. **Color:** Es para indicar el color de la propiedad

Luego el Valor (sirve para indicarle el valor a la propiedad, como el color rojo, verde o tamaño 100px, posición centrada, etc.) Luego de cada valor tiene que terminar con “ ; ” Que pueden ser:

1. RED, GREEN y Otros Colores

1.

\* {

*color*:red;

}

2.

h1 {

*color*:green;

}

3.

.titulo-h2 {

*color*:blue;

}

4.

#element-for {

*color*:pink;

}

5.

[rancio="sii"] {

*color*: brown;

}

5.

h2 b {

*color*: cyan;

}

6.

.h2-class b:hover {

*color*: yellow;

}

**Especificidad en CSS**

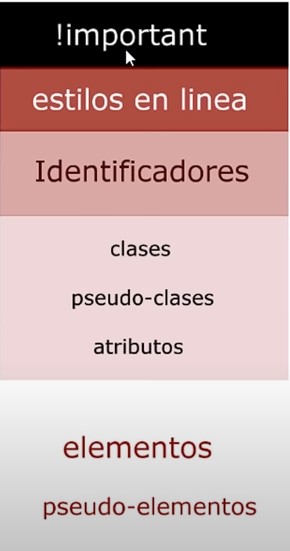
La especificidad es la manera mediante la cual los navegadores deciden qué valores de una propiedad CSS son más relevantes para un elemento y, por lo tanto, serán aplicados.



Al haber 3 opciones para que el navegador elija, siempre elige el que esta último, esto es por un tema de especificidad o jerarquías.

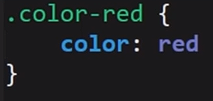
Esto se llama como efecto en Cascada, va eligiendo al que esta último, que tengan la misma jerarquía, al tener diferentes jerarquías, pero mismo atributo, cambia dependiendo la importancia.

Pero la Especificidad va por Jerarquías, por ejemplo:

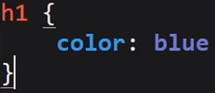


Un ejemplo más practico seria:

Si tenemos un **H1** con **Class**=” color-red” y le queremos poner color rojo:



Y, por último, queremos cambiar el **elemento** todos los **H1** por el color azul:

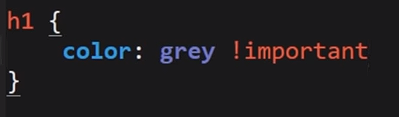


El navegador va a tomar por prioridad a la **CLASE**, ya que la Clase tiene más jerarquías que los **ELEMENTOS.**

**Estilo En Línea:**



**¡IMPORTANT:**

****

**METODOLOGIA BEM**

**Metodología BEM:**

La metodología BEM tiene como objetivo ordenar el código CSS creando clases por cada artículo, como, por ejemplo:

**EN HTML:**

<div class="contact-form">

    <form>

      <input type="text"class="contact-form\_\_input">

      <input type="text"class="contact-form\_\_input">

      <input type="password" class="contact-form\_\_input">

    </form>

</div>

**EN CSS:**

Primer Ejemplo / Utilizamos “.contact-form\_\_input:” para indicar que estamos hablando de la clase de .contact-form / Luego “first-child” para indicar que modifique al primer INPUT.

.contact-form\_\_input:first-child {

*color*: red;

}

Segundo Ejemplo / Utilizamos “—active” u otro.

Esto le indicamos que cuando un atributo tiene –active significa que en CSS modificamos el –ACTIVE.

En HTML:

<input type="text"class="contact-form\_\_input--active">

En CSS:

.contact-form\_\_input--active {

*color*: orange;

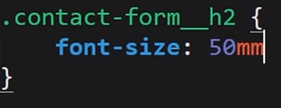
}

**UNIDADES DE MEDIDA EN CSS**

Las unidades de medida que podemos utilizar son:

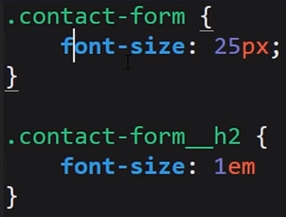
1px / 1mm / 1cm / pt / etc.

Iría en CSS y utilizaríamos la Propiedad “Font-size:”



Estas “**Medidas Relativas**” se utilizan para que se adapten las letras en dispositivos móviles:

**1em**: 16px [default]

El {EM} por defecto es 16px y se va adaptando según la pantalla. Con el Font – size: 25px, le decimos que el {EM} tenga de valor 25px.

Luego otras Medidas Relativas son el “VW” Y “VH”.

.titulo-h2 {

*background-color*:red;

*font-size*: 2em;

*color*: blue;

}

“background-color” Indica el color del fondo de la etiqueta, por ejemplo, un <H2>.

**100vw**: All screen width

*width*: 50vw;

Esto indica que el Ancho tiene que ser la 50vw o la mitad de la Pagina.

**100vh**: All screen height.

*height*: 10vh;

Esto indica que el Largo tiene que ser 10vh

**Propiedades de texto**:

**font-size**: : Tamaño de la fuente.

*font-size*: 1em;

**font-family**: : Tipografía.

*font-family*: Arial;

**line-height**: : Tamaño entre la letra y el tamaño del box.

*line-height*: 2;

**font-weight**: : Grosor de la Letra.

*font-weight*: 500;

**font-stretch**: / **font-variant**: No Se Usan Comunmente.

**Normalize.css**

El normalize.css es un archivo publicado en GitHub que resetea todos los defaults a 0, a este le tenemos que agregar el:

max-width: 100%; al img {} para que en el caso de que se vea una imagen desde un dispositivo móvil se pueda observar en el 100% de la pantalla.

img {

*border-style*: none;

*max-width*: 100%;

}

y agregar también en el {} un box-sizing: border-box:

Esto sirve para indicarle que la caja de una etiqueta tiene que ser de ese tamaño SI O SI.

\* {

*box-sizing*: border-box;

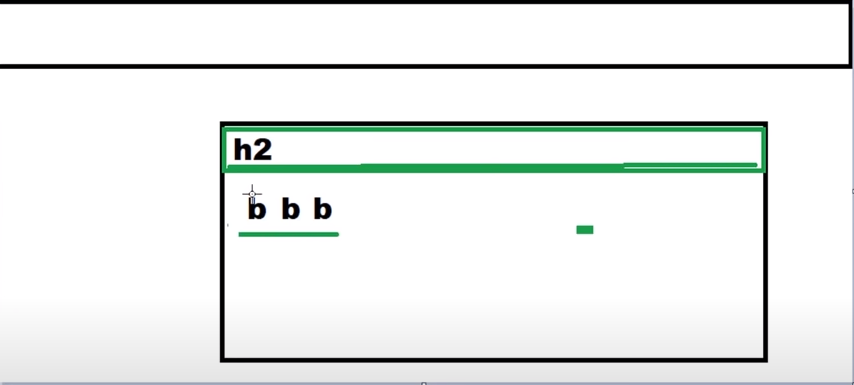
*padding*: 0;

*margin*: 0;

}

**Cajas o Box en CSS**

Las cajas o box, hace referencia a los elementos en bloque y los elementos en línea.



Hay diferentes etiquetas, por ejemplo, el <H2> es un elemento en bloque, ya que, ocupa toda la caja “verde” Este es un elemento en bloque.

Pero esto se puede cambiar, creando una caja dentro de otra caja y que el <H2> poder modificarlo para hacerlo más chico.

Luego los elementos en Línea, como el <b>, <input>, <u>, etc.

Estos son elementos que al terminar el valor que nosotros le añadimos termina ahí y podemos seguir a los lados.

p {

*display*: inline;

*height*: 400px;

*width*: 200px;

}

Con esto indicamos que el <p> sea de *display*: inline; es decir, que sea un elemento en línea, esto tiene como consecuencia, que el *height y width,* no sean **COMPATIBLES** con el elemento en línea.

**Propiedades De Cajas**:

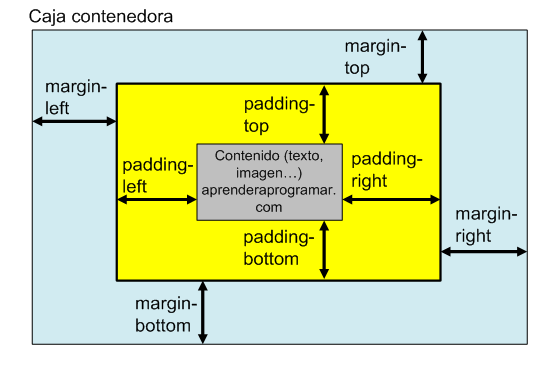
**background-color**: Para añadirle color al fondo.

*background-color*:red;

**display: inline-block**: Sirve para indicarle a la etiqueta, que se comporte como un elemento en línea, pero pueda adquirir las propiedades de un elemento en bloque.

*display*: inline-block;

**Relleno o PADDING:** El padding en **CSS** es una propiedad que mide la distancia entre el texto y la caja definidos, sirve para dar formato y diseño a una página web.



*padding*: 20px;

Esto indica que el “padding”: top/ left/ bottom/ right/. Sean de 20px.

*padding*: 20px 10px 30px 40px;

Esto indica que el “padding”: top 20px/ left 10px/ bottom 30px/ right 40px/.

**BOX-SIZING:**

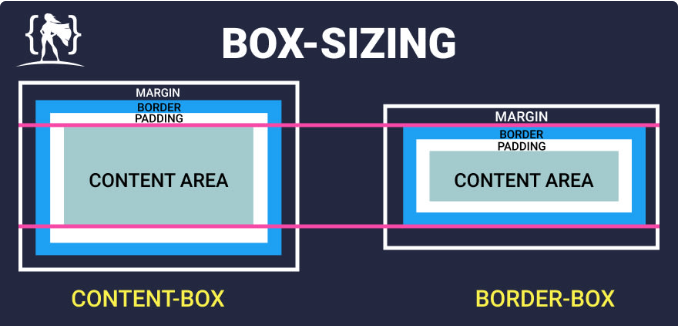
La propiedad box-sizing tiene dos valores: content-box y border-box. El primero es el valor por defecto.

content-box: (tamaño de caja) establece como se calcula el ancho y alto total de un elemento.

*box-sizing*: content-box;

border-box: hace que los elementos mantengan su tamaño y se usa precisamente para que el box model no se rompa.

*box-sizing*: border-box;



**BORDER:**

Border-radius: Esta propiedad sirve para indicar cuanto vamos a “redondear” el borde.

*border-radius*: 30px;

Hay muchos “Border” como, por ejemplo:

*border-color*:;

*border-collapse*:;

pero resumiéndolo podes poner así:

*border*: 5px solid greenyellow;

El borde va a tener 10px y va a ser de estilo solid de color gris.



Hay varios estilos de Bordes, por ejemplo: solid/ dashed/ double/ groove/ridge/ inset/ outset/.

**MARGIN:**

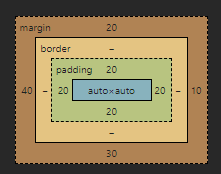
El Margin es la distancia que hay entre las cajas y otros objetos. Sería como que el Margin te deja un espacio entre 2 <h2> por ejemplo.

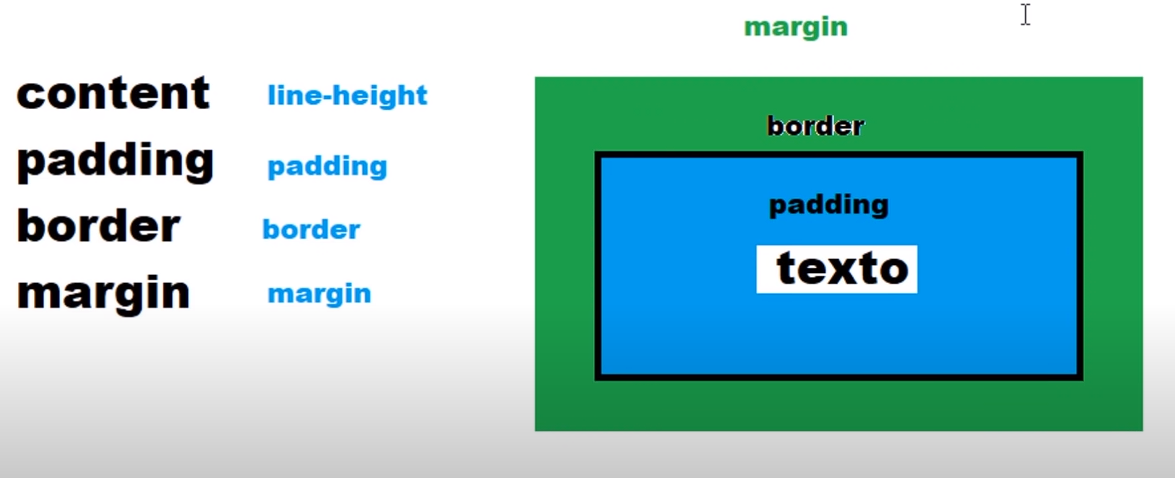
*margin*: 20px;

Esto indica que el “margin”: top/ left/ bottom/ right/. Sean de 20px.

*margin*: 20px 10px 30px 40px;

Esto indica que el “margin”: top 20px/ left 10px/ bottom 30px/ right 40px/.





**BOX CON SOMBRAS:  
  
box-shadow**: La propiedad box-shadow permite proyectar una sombra difuminada desde el marco de casi cualquier elemento.

Lo que se utiliza es:

*box-shadow*: 30px 10px 10px 10px black;

Esto indica que el “box-shadow”: **eje X** 30px/ **eje Y** 10px/ **tamaño** **desenfoque** 10px/ **tamaño del borde** 10px/ **color** black/.



**text-shadow**: cumple la misma función que el box-shadow, pero solo funciona para las letras. Lo que se utiliza es:

*text-shadow*: 2px 2px 7px blue;

Esto indica que el “texto-shadow”: **eje X** 2px/ **eje Y** 2px/ **tamaño** **desenfoque** 2px/ **color** blue/.

Otra Propiedad es el transform: rotate: Esta propiedad sirve para rotar 90 grados, 180 grados, 45 grados, etc. Lo que se utiliza es:

*transform*: rotate(5deg);

Indica que “transform: rotate”: **rotate** / **5 deg**: es que se mueva 5º.

**OUTLINE EN CSS**

La propiedad “outline” en **CSS** se utiliza para dibujar un borde alrededor de un elemento **HTML**, pero a diferencia de los bordes (border) esta propiedad no ocupa espacio dentro del diseño, es decir, no afecta al tamaño de la caja del elemento.

La propiedad **CSS** outline, es una forma reducida para establecer una o más de las propiedades individuales de contorno:

outline-style, outline-width y outline-color en una sola declaración.

Lo que utilizamos en **HTML**:

<div>

    <h2 class="caja1"></h2><h2 class="caja2"></h2>

</div>

Lo que se utiliza en **CSS**:

div h2 {

*width*: 200px;

*height*: 200px;

*background-color*: yellow;

*display*: inline-block;

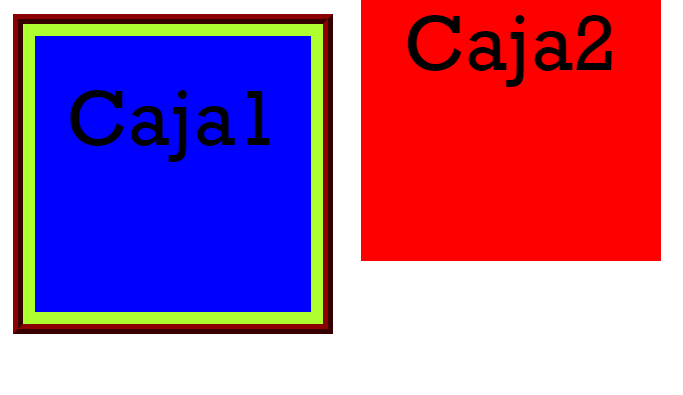
}

Esto indicamos que <div h2> tenga altura y ancho 200px, color de fondo: amarillo y que se desplace como un elemento en bloque.

.caja1 {

*outline*: 4px ridge darkred;

}



-------------------------------------------------------------------------------------------

**Position en CSS**

Position, el posicionamiento predeterminado de cada elemento en **HTML** tiene como valor: “static”, incluso si este valor no ha sido declarado. Todo elemento estático se coloca alineado con la parte superior izquierda en el cuerpo del documento.

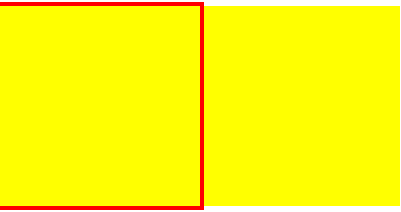
Para cambiar la posición de un elemento, para la propiedad position, son las siguientes: static, relative, sticky,fixed y absolute.

**Position: Relative:**

Atributo “position: relative” el borde de caja1, toma todo el cuadrado.

*border*: 5px solid greenyellow;

*position*:relative;



Relative: Libera al elemento de su posición original y combinado con top, bottom, left y right, nos permite desplazarlo respecto a ella.

.caja1 {

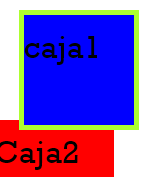
*background-color*: blue;

*position*: relative;

*top*: 10px;

*left*: 25px;

}

El **position relative**, su función es adquirir las propiedades: top/left/right y bottom/.

Sin perder su espacio en pantalla y su punto de referencia es el mismo.

Y añade una nueva propiedad que es el: **Z Index.**

**Z INDEX:**

La propiedad z-index en **CSS** determina el orden de los elementos posicionados en una página web. Un elemento posicionado es aquel que tiene un valor diferente a static en su propiedad position.

Si la “caja1” está por superpuesta o encima de la “caja 2” podes indicar que en la “caja2” este por encima de la caja uno.

Todos por defecto tienen un z index= 0

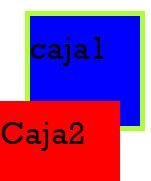
.caja2 {

*background-color*: red;

*position*: relative;

*z-index*: 100;

}

La “caja1” tiene un **top** y **left** con una “position relative”. Que sobrepasa a la “caja2”.

Pero si en la “caja2” añadimos una position relative y un z-index: 100; u otro número.

Lo que pasa es que la “caja2” ahora está por encima.  
El z-index, tiene un problema que se llama el “padre y el hijo”.

Es decir, si hay un <div> con propiedades de **CSS** y dentro de ese <div> hay otro <div> con propiedades de CSS. El “hijo o segundo div” siempre va a estar superpuesta, pero si al <div> hijo le añadimos z-index: -1; no pasa esto

EN **HTML**:

<div class="padre">

    <div class="hijo"></div>

</div>

En **CSS**:

.hijo {

*background-color*: orange;

*position*: relative;

*z-index*: -1;  
-------------------------------------------------------------------------------------------

**Position: Absolute**

El posicionamiento absolute: El Elemento se coloca respecto a su contenedor más cercano, si no encuentra ningún contenedor cercano, el elemento se colocará al **viewport** (El viewport es el área disponible en la pantalla del navegador (no incluye: interfaz del navegador, como botones, barra de menú, etc.) Para que un elemento se vaya al **viewport** hay que darle un “top: 0”

Armar una “caja”: 5 elementos con la propiedad “absolute” adentro 1 en el medio y 4 en cada esquina utilizamos los siguientes códigos:

En **HTML**:

<div class="container">

    <div class="box-1">1</div>

    <div class="box-2">2</div>

    <div class="box-3">3</div>

    <div class="box-4">4</div>

    <div class="box-5">5</div>

 </div>

En **CSS**:

.container {

*position*: relative;

*margin*: 40px auto;

*width*: 500px;

*height*: 300px;

*border*: 4px dashed black;

}

[class^="box"] {

*width*: 140px;

*height*: 80px;

*text-align*: center;

*border*: 4px solid grey;

}

.box-1 {

*position*: absolute;

*background-color*: red;

*top*: 0;

*left*: 0;

}

.box-2 {

*position*: absolute;

*top*: 0;

*right*: 0;

*background-color*: orange;

}

.box-3 {

*position*: absolute;

*bottom*: 0;

*left*: 0;

*background-color*: yellow;

}

.box-4 {

*position*: absolute;

*bottom*: 0;

*right*: 0;

*background-color*: green;

}

.box-5 {

*position*: absolute;

*top*: 0;

*left*: 0;

*right*: 0;

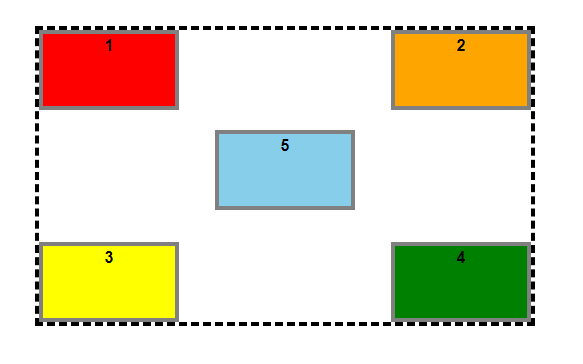
*bottom*: 0;

*margin*: 100px auto;

*background-color*: skyblue;

}

El Resultado es Este:



**Position: Fixed**

“position: fixed” cumple la misma función que un “position: absolute”, lo diferente es que se queda fija, no se mueve de lugar.

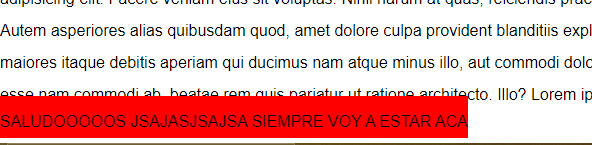
Un ejemplo es en las páginas web cuando tienen un cartel de **publicidades** la publicidad esta siempre fija, aunque nosotros vamos para abajo.

caja-fixed {

*background*: red;

*position*: fixed;

*top*: 10px;

}

Ahora cuando hagamos scroll en la página siempre va a estar ese mensaje.

**Position Sticky:**

position: sticky puede explicarse como una mezcla de position: relative y position: fixed.



-------------------------------------------------------------------------------------------

**Display en CSS**

La función de las cajas proviene de la propiedad display de **CSS**. La propiedad CSS display especifica si un elemento es block o inline.

Los valores básicos son el valor inline y el valor block. El elemento <span> en **HTML**, utiliza el valor de inline mientras que el elemento<div> en **HTML,** utiliza el valor de block.

**Display: Inline**

Los **elementos en línea** no admiten dimensiones (width, height) pero si admiten margin, padding y border, solo ocupan lo que tengan en su contenido.

Si hay varios elementos en línea estarán colocados de izquierda a derecha (uno al lado de otro).

#CAJAS h2{

*display*: block;

}



**Display: Block**

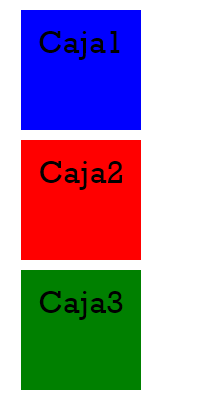
Un **elemento de bloque** ocupa el ancho máximo disponible.

Estos elementos si admiten dimensiones, pero no pueden tener otro elemento a su lado ya que abarcan todo el espacio que tengan disponible por lo tanto estarán colocados de arriba hacia abajo.

#CAJAS h2{

*display*: block;

}



**Display: Inline-Block**

Es una combinación entre los dos tipos de elementos mencionados anteriormente (**block e inline**), los elementos con el valor **inline-block** admiten dimensiones, pero todavía son elementos de línea, es decir estarán colocados uno al lado de otro.

**Overflow En CSS**

La propiedad Overflow en **CSS** nos permite determinar lo que sucede con un elemento cuando su contenido sobrepasa los límites de su contenedor.

* visible: nos permite definir que el contenido que sobresale de su contenedor es visible.

*overflow*: visible;

* hidden:  Con esta opción hacemos que todo el contenido que sobresale de su contenedor se oculte.

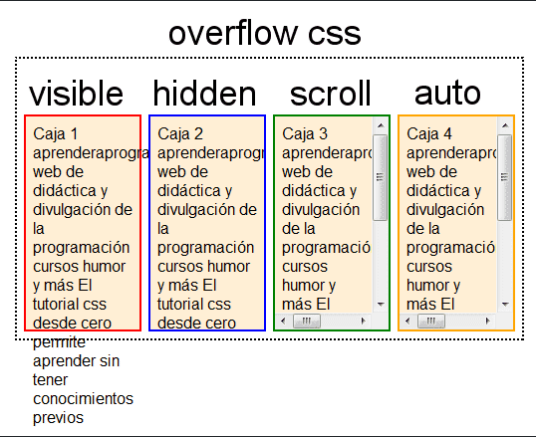
*overflow*: hidden;

* scroll: Las barras de navegación se insertan en el elemento para que pueda acceder al contenido.

*overflow*: scroll;

* auto: Depende del navegador de consulta, algunos agregan barras de navegación y otros no.

*overflow*: auto;



**Float En CSS**

La propiedad **CSS** float ubica un elemento al lado izquierdo o derecho de su contenedor, permitiendo a los elementos de texto y en línea aparecer a su costado.

En CSS:

.textyimg {

*margin*: auto;

*margin-top*: 50px;

*background-color*: gray;

*border*: 4px solid red;

*width*: 50%;

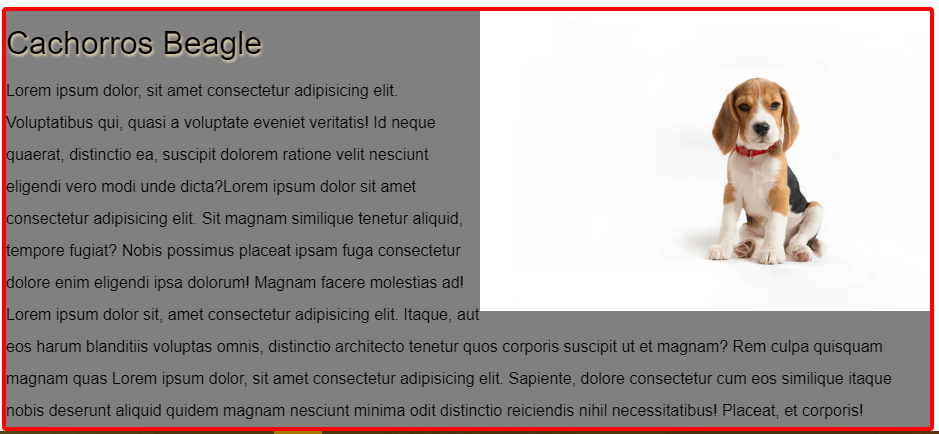
}

.textyimg img {

*height*: 300px;

*float*: right;

}



**PSEUDO-ELEMENTOS EN CSS**

Las propiedades Pseudo-Elementos en **CSS**, Se usa para aplicarle estilo a partes específicas de un elemento, como a la primera letra o línea de un documento.

.pruebadecaja {

*padding*: 20px;

*display*: block;

}

**El pseudo-elemento: first-line:**

El pseudo-elemento ::first-line permite especificar el aspecto de la primera línea de texto. Funciona solo con el **Display: block**.

.pruebadecaja::first-line {

*color*: blue;

*font-size*: 1em;

}

**El pseudo-elemento ::first-letter:**

El pseudo-elemento ::first-letter permite especificar el aspecto de la primera letra de texto. Funciona solo con el **Display: block**.

.pruebadecaja::first-letter {

*color*: blue;

*font-size*: 5em;

}

**El pseudo-elemento ::placeholder:**

Un pseudo-elemento ::placeholder funciona en la entrada de formulario para proporcionar información, en la que el usuario ingrese en ese espacio.

input::placeholder {

*color*: green;

}

**Los pseudo-elementos ::before y ::after**

Contenido generado: **content**

Los pseudo-elementos ::before y ::after permiten añadir contenido a un elemento desde la hoja de estilo, con la propiedad content puede incluir texto, al principio o al final del elemento. El contenido no puede ser seleccionado con el ratón (para copiarlo y pegarlo)

.pseudo::after {

*content*:"PEPESECH";

*color*: green;

*text-decoration*: underline;

}

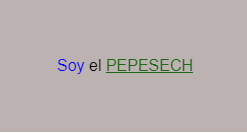
----

.pseudo::before {

*content*:"Soy";

*color*: blue;

}

****

EL “**PEPESECH**” indica after del “**EL**”, es decir después.

Y el “**SOY**” indica before del “**EL**”, es decir antes.

-------------------------------------------------------------------------------------------

**PSEUDO CLASES EN CSS**

Las pseudo-clases sirven para personalizar elementos a los que no se acceden con las clases normales y dependen de una principal.

**Hover:**

Esta pseudo-clase se aplica para el estado de un elemento cuando tienes el cursor encima.

.pseudo-clase {

*height*: 150px;

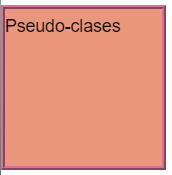
*width*: 150px;

*background-color*: darksalmon; }

.pseudo-clase:hover {

*height*: 250px;

*width*: 250px;

*background-color*: salmon;**** }

Al pasar el cursor por encima se puede apreciar que el Pseudo-Clase HOVER, cambia de color y tamaño.

En **HTML**, se coloca un <div class: “pseud-clase”> luego dentro del <div> coloque un <h1> con el texto Pseudo-clases.

**Link:** Para aplicar estilo a los enlaces que no han sido aún visitados podemos usar: link. De este modo los enlaces se distinguen.

.pseudo-clase:link a{

*color*: red;

*display*: inline-block;

}



**Visited:** Esta pseudo-clase: visited se usa para referirse a los vínculos ya visitados por el usuario.

.pseudo-clase:visited a{

*color*: green;

*display*: block;

}

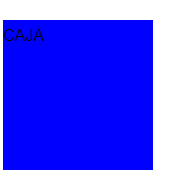


**Active:** Mientras tenemos presionado del botón del ratón sobre un elemento, podemos definir que propiedades tendrá en ese instante.

.active:active{

*background*: blue;

}



**Focus:** El elemento Focus (se encuentra activo). Con la propiedad Focus, se puede cambiar de propiedades. Funciona en los Input.

.contact-form\_\_input:focus {

*background*: grey;

}



**Lang:** Sirve para aplicar estilos en función del idioma de un elemento. En cada elemento se añade una función () que puede ser: (es) / (en).

<b lang="en" class="langs">HELLO HOW ARE YOU? </b>

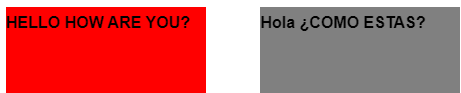
        <b class="langs">Hola ¿COMO ESTAS? </b>

En **CSS**:

.langs:lang(en) {

*background*: red;

}



-------------------------------------------------------------------------------------------

**Object-Fit en CSS**

La propiedad **CSS** object-fit indica cómo el contenido de un elemento reemplazado, por ejemplo, un <img> o <video>, debería redimensionarse para ajustarse a su contenedor.

En **HTML**:

<div class="object-fit">

<imgsrc="https://p4.wallpaperbetter.com/wallpaper/167/925/271/kitten-playful-parquet-wallpaper-preview.jpg">

</div>

EN **CSS**:

.object-fit {

*margin*: 25px;

*height*: 400px;

*width*: 460px;

}

**Object-fit: Contain:**

Aumenta o disminuye el tamaño para llegar al contenedor, conservando la relación de aspecto.

.object-fit img {

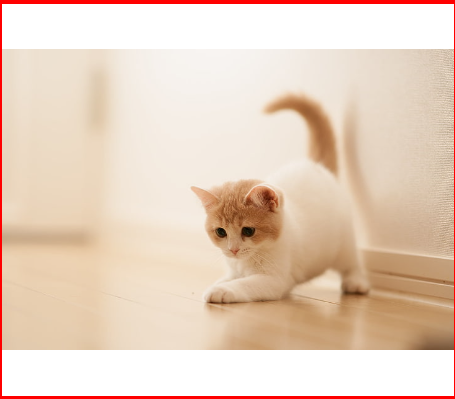
*height*: 100%;

*width*: 100%;

*border*: 4px solid red;

*object-fit*: contain;

}

En la **imagen** se aprecia un borde rojo que indica el tamaño de la “contenedor”.

Pero con Contain, indicamos que la imagen siga manteniendo su tamaño, sin modificar el “contenedor”.

**Object-fit: Cover:**

Se adapta al ancho y altura del contenedor, manteniendo la relación de aspecto. Si la relación de aspecto no coincide con la del contenedor, entonces se realiza un recorte para rellenar el mismo.

.object-fit img {

*height*: 100%;

*width*: 100%;

*border*: 4px solid red;

*object-fit*: cover;

}

Apreciamos que la **imagen** se adaptó en el ancho y altura de los bordes del “contenedor”.

La relación de aspecto se adaptó, recortando partes de la imagen.

**Object-fit: Cover:**

El elemento ignora la altura y el ancho del contenedor, conservando su tamaño original, sin redimensionar el elemento.

.object-fit img {

*height*: 100%;

*width*: 100%;

*border*: 4px solid red;

*object-fit*: none;

}

En la imagen, la altura y ancho del contenedor lo ignoro y conservo su tamaño por defecto. Esto con el **valor**: “none”.

**Object-fit: Scale-Down:**

El elemento se dimensiona como si none o contain se especificaran, para encontrar el tamaño de objeto concreto más pequeño.

.object-fit img {

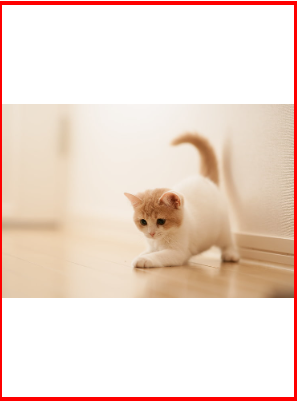
*height*: 100%;

*width*: 100%;

*border*: 4px solid red;

*object-fit*: scale-down;

}

La imagen toma la Escala más baja para adaptarse dentro del “contenedor”.

Si la imagen es demasiado grande solo tomara una parte de la imagen. El valor es **scale-down**.  
**Object-fit: Fill: (POR DEFECTO)**

Cambia el elemento para que se ajuste al contenedor, si la relación de aspecto no coincide, este se estira para rellenar todo el contenedor.

.object-fit img {

*height*: 100%;

*width*: 100%;

*border*: 4px solid red;

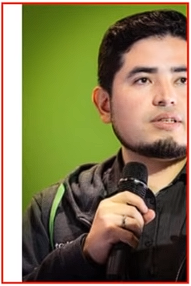
*object-fit*: fill;}

El valor: “**Fill**” es el que viene por defecto cuando ponemos una imagen o video en **HTML**. Se ajusta al contenedor que está dentro y se estira si la relación de aspecto no coincide.

**Object-Position:**

La propiedad **CSS** object-position especifica la alineación del contenido del elemento reemplazado dentro de la caja del elemento. (Las áreas de la caja que no estén cubiertas por el objeto del elemento reemplazado mostrarán el fondo del elemento).

*object-position*: top / left / bottom / right;

Esta imagen esta corrida hacia la derecha “right” y tiene “1em” desplazado del “contenedor”.

*object-position*: 1em / 1px / 100%;

-------------------------------------------------------------------------------------------

**CURSOR EN CSS**

Permite cambiar el modelo y estilo de un cursor. Pueden ser de esperando, texto, pulsar, zoom, flechas, etc. Para cambiar el cursor se pone la propiedad:

*cursor*: pointer;

(El cursor pointer es el de pulsar)

La siguiente página muestra los diferentes cursores que hay para añadir a nuestra página web: <https://www.w3schools.com/cssref/tryit.php?filename=trycss_cursor>

-------------------------------------------------------------------------------------------

**COLORIZACION EN CSS**

Usar colores en formatos específicos tiene una ventaja por sobre utilizar los colores predefinidos con nombres. Ya que, los distintos navegadores pueden interpretar dichos colores con una tonalidad diferente a la que queremos.

Existen varios sistemas de colorización: Hexadecimal, RGB, RGBA, HSL, HSLA, etc.

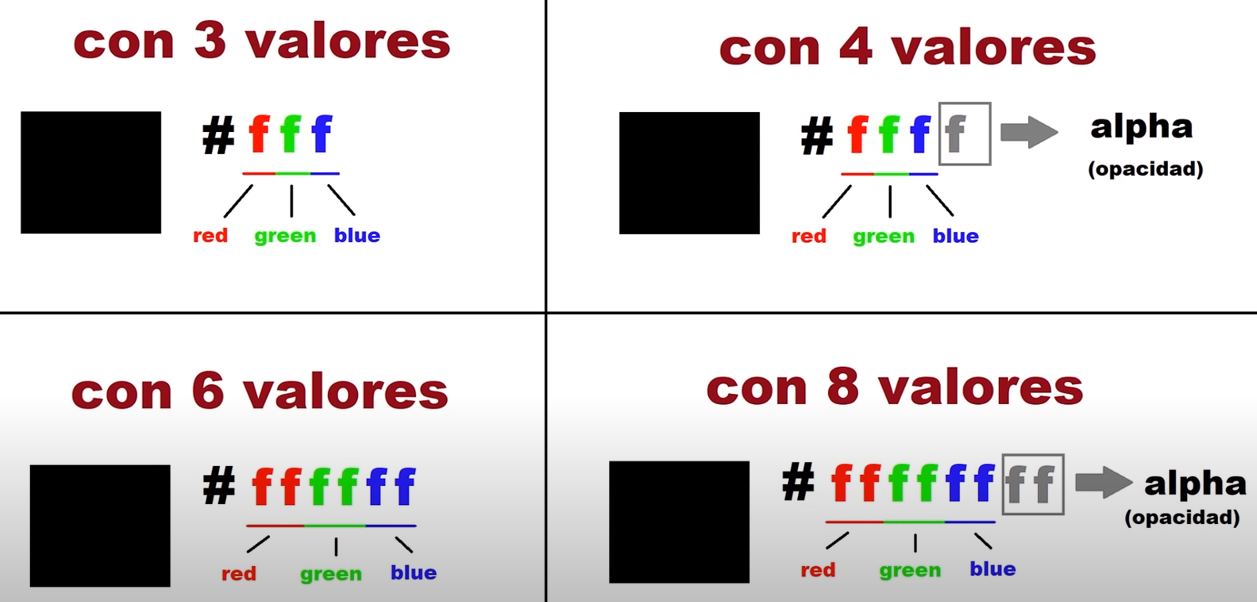
Se puede indicar con el **nombre del color**: Rojo / Azul /Verde.

**Hexadecimal:**

Sistema de 16 unidades. Del 0 al 10: 0-10 / Del 11 al 16: A-F

Ejemplo:

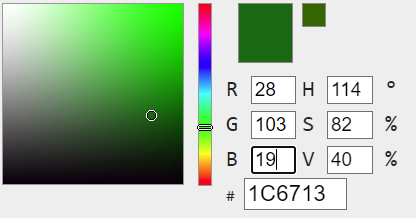
#0000FF / #FFFF00 / #008000 / **#000000**

****

**RGB:**

Red Green Blue Se declaran 3 valores del 0 al 255 (intensidad del color).

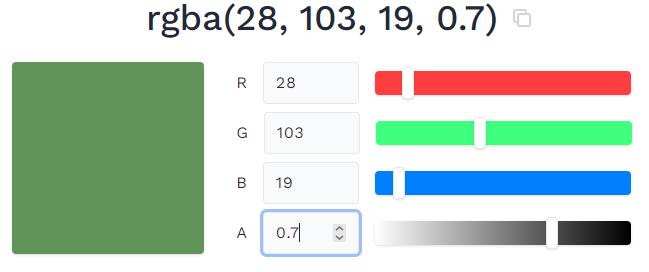
rgb(28, 103, 19)



**RGBA:**

Igual que RGB, pero se le añade el valor de la opacidad (0 al 1) A=Alpha

rgba(28, 103, 19, 0.7)



**RESPONSIVE DESIGN Y MOBILE FIRST EN CSS**

La función del **responsive design** es adaptar el diseño de la página web para todas las resoluciones.

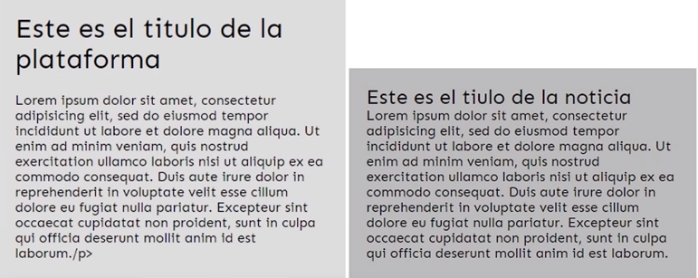
**Mobile first:**

Empezar a crear la web especialmente para resoluciones pequeñas (celulares, tablets, etc.), y posteriormente adaptarla para escritorio. Google recomienda las páginas con este concepto.

Se trabaja con media-query: Cambiar el estilo de los elementos a partir de cierta resolución.

Añadir etiqueta al head del **HTML**: (OBLIGATORIO)

<meta name="viewport” content="width=device-width, initial-scale=1.0">

Si tenemos 2 títulos y párrafos ordenados con display: inline-block

En Tablet y celulares la resolución es diferente que, en la computadora, entonces los de celular verían mal la página.

Por eso se pone la propiedad:

@media only screen and (*max-width*:800px)

Esto indica: “Cuando la pantalla indica resoluciones menores de 800px de ancho” va a hacer algo podríamos poner que los <div> sean con un display: block y un width: 100%.

En **CSS**:

@media only screen and (*max-width*: 800px) {

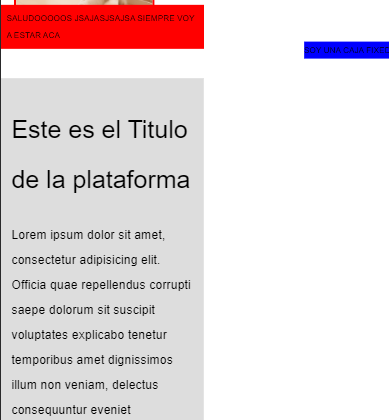
[class^="cajita"] {

*display*: block;

*width*: 100%;

    }

}

Es decir, cuando el ancho de la resolución sea menor a 800px, Las “cajitas” se colocarán uno debajo de otro.

Si inspeccionamos la pagina y ponemos la vista de un celular, por ejemplo: Phone 12.

Se observaría de esta forma y con esto la página web estaría adaptada para dispositivos móviles.

-------------------------------------------------------------------------------------------

**MENU RESPONSIVE EN CSS**

Menú que se despliega a partir de determinada resolución. Se oculta el <nav> normal y se muestra otro adaptado para celulares.

**Paso 1:**

Se crearon 2 navs, uno para resolución de escritorio y el otro responsive. En el responsive se agregan 2 divs: Botón y Contenedor del menú



**Paso 2:**

Pegar en el head del **HTML** el siguiente código:

<script src="https://kit.fontawesome.com/62ea397d3a.js" crossorigin="anonymous"></script>

Con este script se tendrá acceso a íconos transparentes y modificables en color.

**Paso 3:**

Añadir los íconos mediante las clases, por ejemplo:

<li class="nav\_\_li"><i class="fas fa-home"></i><a href="#"> Inicio </a></li>

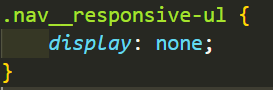
<li class="nav\_\_li"><i class="fas fa-user-friends"></i><a href="#"> Sobre nosotros </a></li>

<li class="nav\_\_li"><i class="fas fa-dollar-sign"></i><a href="#"> Gana dinero </a></li>

<li class="nav\_\_li"><i class="fas fa-question-circle"></i><a href="#"> Ayuda </a></li>

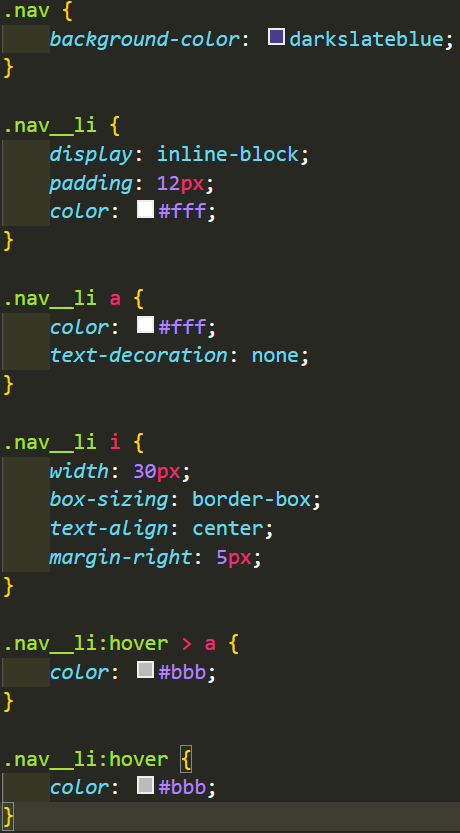


**Paso 4:**

En **CSS**:

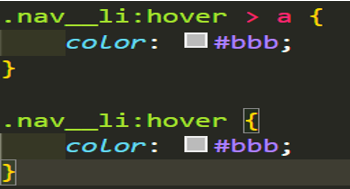
Se oculta el nav responsive.

Luego se modifica en otro lado.

Acá modificamos la propiedad del <nav>.

Primero modificamos el color de fondo, luego que sean todos en línea con inline-block, que el “li” no tenga decoración y un color, padding y margin.

Texto en el centro y colores.

 Añadimos “Hover” al pasar el mouse por encima se cambia de color.



**Paso 5:**

Se declara el evento en el cual, si el ancho de la resolución es menor a 500px: Se oculta el nav de escritorio y aparece el nav responsive

@media only screen and (*max-width*:500px) {

    .nav\_\_ul {

*display*: none;

    }

    .nav {

*height*: 37px;

*width*: 100%;

*background-color*: #69e;

    }

    .nav\_\_responsive-ul {

*display*: block;

*position*: absolute;

*width*: 100%;

*background-color*: #7af;

*margin-top*: 37px;

    }

    .nav\_\_responsive-li a {

*color*: #016;

*text-decoration*: none;

    }

}



Cuando la resolución llega a 500px de ancho (Resoluciones que solo tienen celulares y Tablet).

Tienen diferentes <navs> uno para resoluciones más anchas y otra para resoluciones menores a 500px.

-------------------------------------------------------------------------------------------

**DISPLAY FLEX EN CSS**

Flexbox = Cajas flexibles

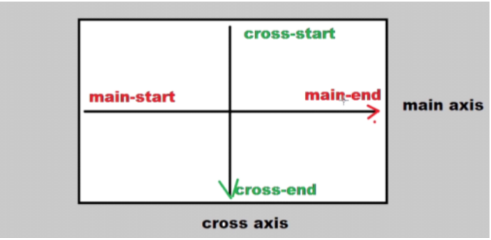
Su función es maquetar sitios web. Nació como una alternativa simplificada de las tablas.

• Flex requiere de dos elementos: Flex container y Flex ítems.

• De por sí, el contenedor se comporta como un block.

• Los hijos de un flex container son flex ítems, pero no sus nietos.

Si hay un desbalance en la cantidad de texto en cada ítem: No cambian las alturas ni los widths, se ajusta el contenido.



Main axis = Eje X - De izquierda a derecha.

Cross axis = Eje Y - De arriba abajo.

(Tiene una dirección en la que apunta.)

Ejemplo en **CSS**:

.flex-container {

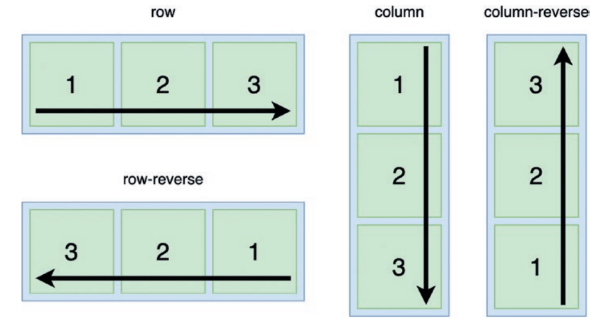
*display*:flex

}

**Flex-Direction:**

Nos permite cambiar la dirección del Main-axis. La propiedad se le aplica al contenedor. El valor por defecto es "row".

|  |  |  |
| --- | --- | --- |
| **row** | Ordena los flex-ítems en FILAS | De izquierda a derecha |
| **row-reverse** | Ordena los flex-ítems en FILAS (al revés | De derecha a izquierda |
| **Column** | Ordena los flex-ítems en COLUMNAS | De arriba a abajo |
| **Column-reverse** | Ordena los flex-ítems en COLUMNAS | De abajo a arriba |



**Flex-wrap:**

Mantiene el width o height de las cajas en una resolución menor. Pone las cajas una por debajo de otra.

Valores: Nowrap (por defecto) / Wrap / Wrap-reverse (al reducir la resolución, la última caja se pone arriba y así sucesivamente).

**Flex-flow:**

Shorthand de flex-direction y flex-wrap Ejemplo:

.flex-container {

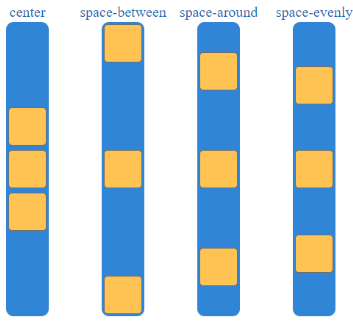
*display*:flex;

*flex-flow*: column wrap;

}

**Justify-content:** Valores:

|  |  |
| --- | --- |
| **center** | Las cajas se centran al medio de la pantalla. |
| **space-around** | Márgenes iguales --> Lo mismo que margin: auto. |
| **space-between** | Calcula un margen automático para todas las cajas. Quiere que todas las cajas tengan la mayor y la misma distancia entre sí. |
| **space-evenly** | Calcula que las distancias entre las cajas sean exactamente iguales. |



**Align-items:**

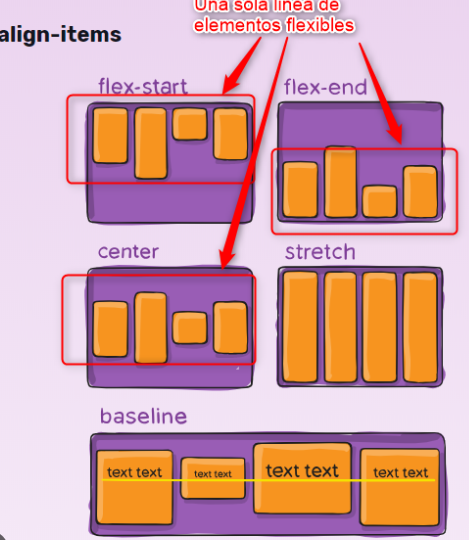
Valores: Se aplica cuando solo hay 1 línea de flex-ítems.

|  |  |
| --- | --- |
| **center** | Las cajas se centran verticalmente en la pantalla visible. |
| **flex-start** | Las cajas se alínean al comienzo de la pantalla visible. |
| **flex-end** | Las cajas se alínean al final de la pantalla visible. |
| **stretch** | Por defecto (si el vh o vw es 100, las cajas se estiran hasta el final de la pantalla visible). |
| **baseline** | Todas las cajas se van para abajo.  A medida que se reduce la resolución algunas cajas se van yendo gradualmente hacia arriba. |

**Align-content:**

Se aplica cuando se trabaja con varias líneas de flex-items.

• Tiene los mismos valores que Align-items.



**Propiedades Flex-items:**

Al igual que el flex container, los flex items también cuentan con distintas propiedades reservadas.

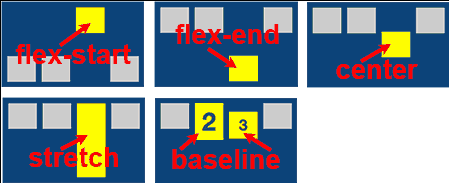
Entre ellas se encuentran:

**Align-self:**

Determina la alineación en el cross-axis de determinada caja de manera independiente.

*align-self*: center;

|  |  |
| --- | --- |
| **center** | La caja se centra verticalmente en la pantalla visible. |
| **flex-start** | La caja se alínea al comienzo de la pantalla visible. |
| **flex-end** | La caja se alínea al final de la pantalla visible. |
| **stretch** | Por defecto (si el vh o vw es 100, la caja se estira hasta el final de la pantalla visible) |
| **baseline** | La caja se va para abajo. Si se reduce la resolución, en determinado punto la caja se iría para arriba de las demás. |



**Margin:**

El funcionamiento del margin en las cajas flexibles es un poco diferente al de las “cajas” normales.

Por ejemplo, si le proporcionamos un margin-left: auto, la caja se va completamente hacia la derecha, y viceversa.

**Flex Grow:**

Reparte el espacio sobrante entre determinadas cajas. Es decir, los flex-items elegidos se van a estirar hasta ocupar todo el ancho de la pantalla en partes iguales, y si se reduce la resolución, se quedan hasta el min-width.

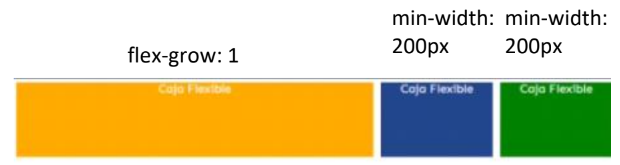
Es recomendable aplicar un flex-wrap: wrap, para que las cajas se coloquen una debajo se otra cuando se llegue a determinada resolución. (min-widths sumados de cada caja + margins).

Ejemplo:

.flex-item {

*flex-grow*: 1;

}



En el caso de la imagen, el ancho de la página se repartiría en:

• Las 2 cajas con min-width

• La caja con ancho ilimitado (flex-grow: 1).



**Flex-basis:**

Similar al width. En términos de especificidad, el flex-basis está por encima del width. Al tener más relevancia en los rangos jerárquicos del entorno de flex, se recomienda usarlo por sobre width, ya que cumplen la misma función.

**Flex-shrink:**

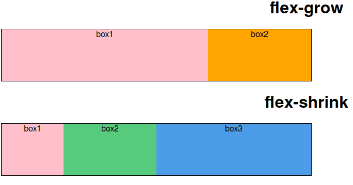
Determina qué cantidad de espacio va a ceder el flex-item cuando tenemos una resolución inferior a la deseada, en términos relativos (proporciones). Por ejemplo:

• Hay 2 cajas con flex-shrink: 2, y hay 1 caja con flex-shrink: 1

• Pero queremos que todas las cajas midan 300px en algún punto.

Entonces, lo que pasaría sería lo siguiente: La caja con shrink 2 cedería el doble de espacio que las otras dos hasta que **TODAS** lleguen a 300px de ancho.

El valor por defecto es 1 / / 0 = No cede espacio.



**Flex (shorthand):**

Agrupa las tres propiedades anteriores: flex-grow, flex-shrink y flex-basis. El primer parámetro es obligatorio, es decir, flex-grow.

Por ejemplo:

.red {

*flex*:1 0 350px;

}

**Order:**

Similar al z-index. Es decir, el flex-item que tenga mayor valor será el que esté posicionado más hacia la dirección en la que apunta el main axis. (Por defecto hacia la derecha, pero también podría ser hacia la izquierda, arriba o abajo, en caso de que esté estipulado así).

Al igual que en el z-index, se recomienda aumentar los valores por cada ítem de 50 en 50 o de 100 en 100 para dejar espacios en el aire, en el caso de que se quieran agregar más en otros elementos.

-------------------------------------------------------------------------------------------

**DISPLAY GRID EN CSS**

Grid = grilla

Al igual que Flexbox, es una propiedad de display. Es un estilo de layout en el cual se trabaja con grillas. Una caja grid funciona a través de celdas, rows y columns (tracks), áreas, etc.

• Requiere de una serie de elementos: Grid-container y Grid-items

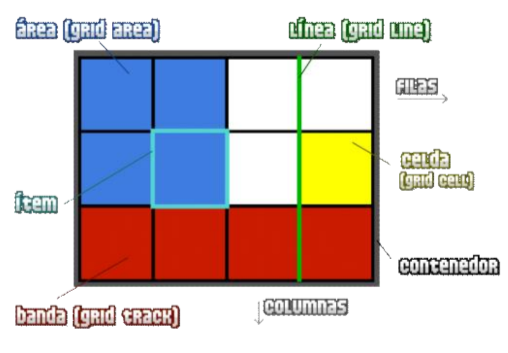
• El contenedor se comporta como block

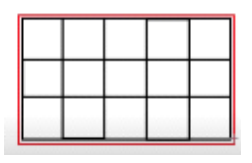
• Los hijos directos del Grid-container son Grid-items, sus nietos no.

.grid-container {

*display*: grid;

}



**Grid container:**

Es la totalidad del contenedor.

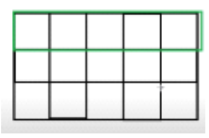
**Grid item:**

Los hijos directos de un grid-container. No son necesariamente celdas (grid-cell).

**Grid cell:**

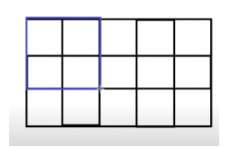
Cada una de las celdas, es decir, las divisiones que se forman dentro del container.

**Grid track:**

Existen dos tipos: column y row.

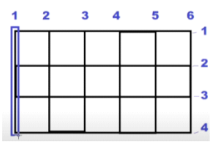
Columns + Rows = Grid Tracks

**Grid area:**

Determinadas áreas dentro del contenedor que seleccionamos para darles propiedades en específico. No están definidas por defecto, es decir, se definen manualmente.

Las áreas deben estar obligatoriamente conformadas por un grupo de celdas consecutivas, tanto vertical como horizontalmente. Por lo tanto, no se pueden agrupar celdas en diagonal ni de ninguna otra manera que no forme un cuadrilátero.

**Grid line:**

Existen dos tipos: column-line y row-line

Column-lines = Columns + 1

Row-lines = Rows + 1

**Grid-template-rows:**

Determina la cantidad de filas. Como parámetros se colocan las medidas para cada fila. (ya sean medidas exactas o relativas)

Medidas exactas (estáticas):

.grid-container {

*display*: grid;

*grid-template-rows*:150px 150px 150px;

}

Medidas relativas (proporcionales):

1fr = 1/total

**Grid-template-columns:**

Comparte las mismas características que Grid-template-rows.

.grid-container {

*display*: grid;

*grid-template-columns*: 1fr 2fr 1fr;

}

La segunda columna tendría el doble de alto que las otras filas.

**Grid-column-gap:**

Determina la distancia entre las columnas.

**Grid-row-gap:**

Determina la distancia entre las filas.

**Grid-gap:**

Propiedad shorthand. Determina la distancia entre una celda y la otras respectivamente.

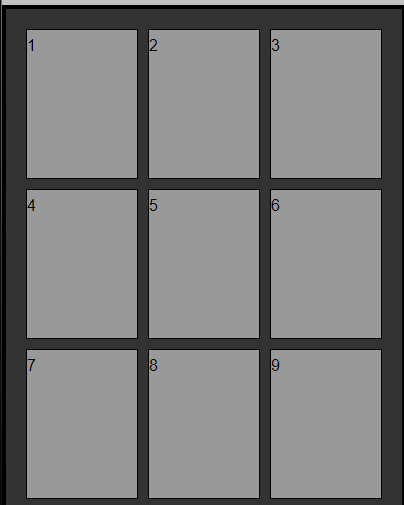
(Fusión de las propiedades grid-column-gap y grid-row-gap)

1. Distancia entre la celda y las demás celdas.

.grid-container {

*grid-gap*: 10px;

}



**Grid-row:**

Aplica lo mismo que para la propiedad grid-column.

**Grid-column:**

Define la cantidad de celdas que va a ocupar un grid-item.

Ejemplo:

.grid-item:first-child {

*background*: #900;

*grid-column*: 1/2;

*grid-row*: 1/ span 2;

}

(El grid-item va a ocupar el espacio: la línea vertical 1 hasta la 3).

Esto genera por "**inercia**", que las demás celdas se empujen sucesivamente, y si fuese el caso se crearían nuevas filas para incluir a los elementos desplazados.

Segundo Ejemplo:

.grid-item:nth-child(2) {

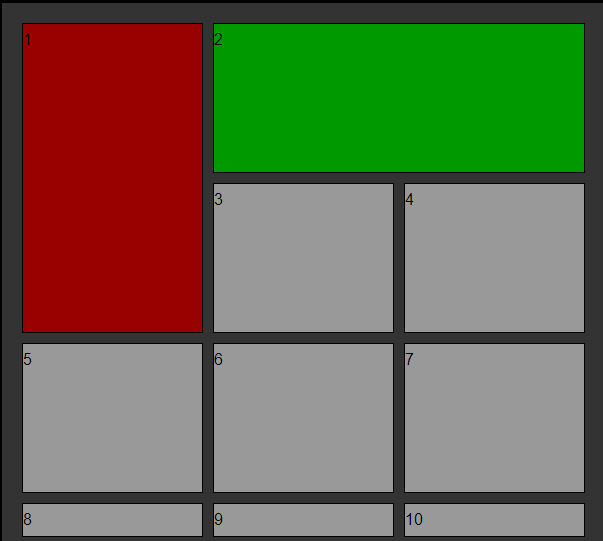
*background*: #090;

*grid-column*: 2/4;

}

Span = indica la cantidad de columnas a ocupar.

(Va a ocupar 2 columnas desde la línea vertical 1).



**Repeat()**

Función que indica la cantidad de columnas o filas a colocar, y sus

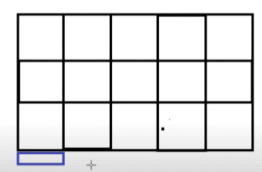
respectivas medidas. Ejemplo:

*grid-template-rows*: repeat(3,150px);

3 columnas de 150 px.

**Grid implícito y explícito:**

Se entiende como Grid Implícito a aquellas celdas "sobrantes", que no han sido declaradas expresamente en el código. Ya sea porque eventualmente ciertas celdas fueron desplazadas forzadamente generando esta situación, o porque simplemente no fueron declaradas. Es decir, no forman parte del Grid Explícito.



Además, los elementos que formen parte del Grid Implícito cuentan con algunas propiedades particulares.

**Grid-auto-rows:**

Su función es gestionar y definir el tamaño de las celdas que eventualmente vayan a formar parte del Grid Implícito. En este caso, le decimos al container que, en caso de existir de celdas sobrantes, estas pasarán a formar nuevas **FILAS** y no columnas.

Ejemplo:

.grid-container {

*display*: grid;

*grid-auto-rows*: 150px;

}

**Grid-auto-columns:**

Mismo funcionamiento que Grid-auto-rows, pero en caso de existir celdas sobrantes, estas pasarán a formar nuevas **COLUMNAS** y no filas.

**Grid-auto-flow:**

Gestiona los espacios vacíos en la **grilla** en alguna situación en particular que genere esto.

|  |  |
| --- | --- |
| **Dense** | Ocupa los espacios libres de manera forzada, adapta  determinadas celdas para que ocupen esos lugares. |

**Grid dinámico**

Son propiedades que le añaden ciertas funcionalidades con respecto a la distribución de las celdas en la grilla. Por defecto, el ancho mínimo de una celda radica en el ancho de la palabra más larga. (En cualquier unidad de medida, ya sea exacta o relativa).

**Min-content:**

El ancho de la celda se adapta a la cantidad mínima de contenido (sin cortar las palabras), es decir, hasta el final de la palabra más larga.

.grid-container {

*grid-template-rows*: repeat(3, min-content);

}

**Max-content:**

El ancho de la celda se adapta a la cantidad máxima de contenido. (Todo el texto de corrido en 1 línea).

**Minmax()**

Determina la medida mínima y máxima de ciertas filas o columnas.

Ejemplos:

.grid-container {

*display*: grid;

*grid-template-rows*: repeat(3, minmax(100px, 1fr));

}

.grid-container {

*display*: grid;

*grid-template-rows*: repeat(3, minmax(min-content, max-content));

}

**Auto-fill:**

Calcula la cantidad máxima de columnas o filas posibles con determinadas propiedades.

Ejemplo:

.grid-container {

*display*: grid;

*grid-template-rows*: repeat(auto-fill, minmax(100px, 1fr));

}

Es decir, la cantidad máxima de columnas posibles con mínimo 100px y 1fr máximo. (Puede dejar espacios vacíos).

**Auto-fit:**

Ajusta las columnas para ocupar todo el espacio disponible, sin dejar espacios vacíos.

**Alineación y Control de Flujo:**

Existen diversos tipos de alineación en Grid:

|  |  |
| --- | --- |
| **Alineación general** | Todos los elementos |
| **Alineación parcial** | Columnas o filas |
| **Alineación particular** | Elemento individual |

Propiedades:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**ITEMS**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**Justify-items**

Similar al justify-content de Flexbox. Alínea los ÍTEMS horizontalmente.

|  |  |
| --- | --- |
| **center** | Centra las cajas y el texto |
| **start** | Alínea las cajas para la IZQUIERDA del contenedor y centra el texto. |
| **end** | Alínea las cajas para la DERECHA del contenedor y centra el texto. |

Ejemplo:

.grid-container {

*justify-items*: center;

}

**Align-items**

Alínea los ÍTEMS verticalmente.

|  |  |
| --- | --- |
| **center** | Alínea las cajas en el CENTRO del contenedor. |
| **start** | Alínea las cajas para ARRIBA en el contenedor |
| **end** | Alínea las cajas para ABAJO en el contenedor |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**CONTENT**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Justify-content**

Alínea las COLUMNAS o FILAS horizontalmente.

|  |  |
| --- | --- |
| **center** | Centra las filas y columnas |
| **start** | Alínea las filas y columnas hacia la IZQUIERDA. |
| **end** | Alínea las filas y columnas hacia la DERECHA. |
| **space-around** | Márgenes iguales --> Lo mismo que margin: auto |
| **space-between** | Calcula un margen automático para todas las cajas. Quiere que todas las cajas tengan la mayor y la misma distancia entre sí. |
| **space-evenly** | Calcula que las distancias entre las cajas sean exactamente iguales. |

**Align-content**

Alínea las COLUMNAS o FILAS verticalmente

|  |  |
| --- | --- |
| **center** | Centra las filas y columnas |
| **start** | Alínea las filas y columnas hacia ARRIBA. |
| **end** | Alínea las filas y columnas hacia ABAJO. |
| **space-around** | Márgenes iguales --> Lo mismo que margin: auto |
| **space-between** | Calcula un margen automático para todas las cajas. Quiere que todas las cajas tengan la mayor y la misma distancia entre sí. |
| **space-evenly** | Calcula que las distancias entre las cajas sean exactamente iguales. |

**Aplicado al Grid-item:**

Propiedades reservadas que se aplican individualmente para gestionar determinado Grid-item.

**Align-self:**

Alínea individualmente un Grid-item en el EJE Y (verticalmente)

|  |  |
| --- | --- |
| **start** | Alínea las cajas lo máximo posible para ARRIBA (hasta que se tope con el margin de otra caja) |
| **end** | end Alínea las cajas lo máximo posible para ABAJO (hasta que se tope con el margin de otra caja) |

.grid-item: nth-child(8) {

*align-self*: start;

}

**Justify-self:**

Alínea individualmente un Grid-item en el EJE X (horizontalmente)

|  |  |
| --- | --- |
| **start** | Alínea las cajas lo máximo posible para IZQUIERDA (hasta que se tope con el margin de otra caja) |
| **end** | end Alínea las cajas lo máximo posible para la DERECHA (hasta que se tope con el margin de otra caja) |

**Place-self:**

Shorthand de Align-self y Justify-self

Parámetros: Align-self / Justify-self

.grid-item: nth-child(8) {

*place-self*: start end;

}

**Stretch:** (default)

**Order:**

Reemplaza la posición de determinado Grid-item por otro en específico.

.grid-item: nth-child(6) {

*order*: 3;

}

(El Grid-item 2 se desplazará hasta la posición del 3 y viceversa).

**Grid Área:**

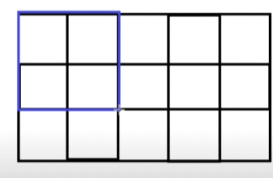
Un Grid-area es un conjunto consecutivo de Grid-cells.

Son determinadas áreas dentro del contenedor que seleccionamos para darles propiedades en específico. No están definidas por

defecto, es decir, se definen manualmente.

Las áreas deben estar obligatoriamente conformadas por un grupo de celdas consecutivas, tanto vertical como horizontalmente.

Por lo tanto, no se pueden agrupar celdas en diagonal ni de ninguna otra manera que no forme un cuadrilátero.



**Grid-template-areas:**

Se asigna la distribución de las áreas en relación a lo que se estipule en el texto.

En la misma línea, si se repite 1 vez un área "aside" y 2 veces un área "Main":

El área "aside" tendrá 1/3 del ancho y el área "Main" tendrá 2/3.

*grid-template-areas*:

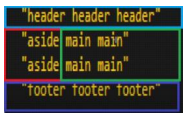
    "header header header"

    "aside main main"

    "aside main main"

    "aside main main"

    "footer footer footer";



(Se crearon 4 Grid-areas: header, aside, main y footer)

**Asignar áreas a los Grid-items:**

.grid-header {

*background*: #f96;

*grid-area*: header;

}

.grid-main {

*background*: #96f;

*grid-area*: main;

}

.grid-aside {

*background*: #888;

*grid-area*: aside;

}

.grid-footer {

*background*: #6f9;

*grid-area*: footer;

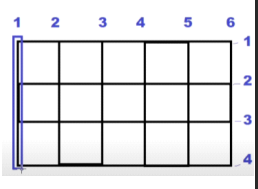
}

**Grid Lines:**

Las Grid Lines son aquellas líneas divisoras que se ubican entre las columnas y filas. Existen dos tipos: column-line y row-line

• Column-lines = Columns + 1

• Row-lines = Rows + 1



**Nombrar Grid Lines:**

.grid-container {

*grid-template-rows*:

    [f-line]

    150px

    [s-line]

    150px

    [t-line]

    150px

    [h-line]

    ;

*grid-template-columns*: repeat(3, 150px);

}

.grid-item:nth-child(2) {

*grid-row*: s-line / h-line;

}

(El segundo Grid-item va a ocupar los espacios desde la segunda hasta la cuarta línea divisora).

**Grid Shorthand:**

**Grid-template**

Shorthand de Grid-template-columns y Grid-template-rows.

Ejemplo:

.grid-container {

*grid-template*: repeat(3, 150px) / repeat(4, 200px);

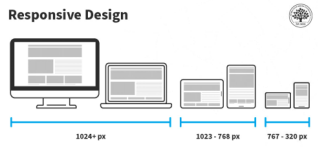
}

(3 columnas de 150px y 4 filas de 200px).

-------------------------------------------------------------------------------------------

**Responsive Design EN CSS**

El Responsive Design es una metodología / técnica utilizada para adaptar todo el contenido de una página web a distintas resoluciones, con el objetivo de optimizar la experiencia de usuario (UX).



• Se trabaja con:

○ **Estructuras flexibles** (contenedores, imágenes y videos flexibles)

○ **Media-queries**

• **@Media** = Condicional que consulta y valida las características del dispositivo que accede a la página:

○ Ancho y alto de la ventana gráfica

○ Ancho y alto del dispositivo

○ Orientación (tablets, en modo horizontal o vertical)

○ Resolución (FHD, QHD, etc.)

• **Tipos de Media Query:**

|  |  |
| --- | --- |
| **all** | Apto para todos los dispositivos |
| **print** | Destinado a material impreso / Modo vista previa de impresión |
| **screen** | Destinado a las pantallas |
| **speech** | Destinado a sintetizadores de voz |

• **Operadores**: AND, OR

• **Orientation:** landscape (horizontal), portrait (vertical).

**Metodologías:**

**Mobile First (recomendado)**

Empezar a crear la web especialmente para resoluciones pequeñas (celulares, tablets, etc.), y posteriormente adaptarla para escritorio o resoluciones mayores.

• Google recomienda las páginas con este concepto.

• Otorga ventajas en el ámbito del SEO (posicionamiento en navegadores)

• Es una buena práctica ya que facilita distribuir los elementos.

**Desktop First:**

Empezar la web orientada a resoluciones grandes, por ejemplo, monitores normales

(16:9) o monitores ultra-wide (21:9), y posteriormente adaptarla a resoluciones menores.

**Content First:**

Estructurar la web en base al contenido, independientemente de si va a estar orientada para resoluciones grandes o pequeñas.

**Sintaxis Media Queries:**

1. Regla / 2. Tipo / 3. Condicional / 4. (Parámetro: valor)

@media screen and (*max-width*: 400px) {

display: none;

}

@media screen and (*min-width*: 400px) and (*max-width*: 650px) {

    display: block;

    background-color: violet;

}



**Práctica Display Grid y Flex en CSS**

EN **HTML**:

<body>

    <div class="content">

        <header class="header">Header</header>

        <article class="main">Main</article>

        <aside class="aside">Aside</aside>

        <footer class="footer">Footer</footer>

    </div>

</body>

EN **CSS**:

\* {

*font-family*: sans-serif;

}

.content {

*display*: flex;

*flex-direction*: column;

*height*: 100vh;

}

.header, .main, .aside, .footer {

*padding*: 20px;

}

.header {

*background-color*: lightblue;

*flex-basis*: 60px;

}

.main {

*background-color*: slateblue;

*flex-basis*: 300px;

*flex-grow*: 2;

*flex-shrink*: 0;

}

.aside {

*background-color*: darkslateblue;

*flex-basis*: 200px;

*flex-grow*: 1;

*flex-shrink*: 0;

}

.footer {

*background-color*: violet;

*flex-grow*: 1;

}

@media screen and (*min-width*: 700px) {

.content {

*display*: grid;

*grid-template-rows*: 60px 1fr 1fr 1fr 1fr 80px;

*grid-template-columns*: repeat(3, 1fr);

}

.main {

*grid-row*: 2 / 6;

*grid-column*: 1 / span 2;

}

.aside {

*grid-row*: 2 / 6;

}

.header {

*grid-column*: 1 / span 3;

}

.footer {

*grid-row*: 6 / 7;

*grid-column*: 1 / span 3;

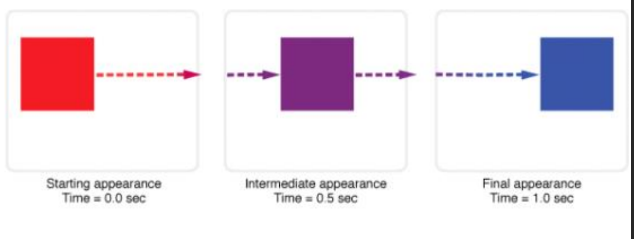
}

}

-------------------------------------------------------------------------------------------

**TRANSITION EN CSS**

Propiedad que permite realizar cambios graduales en los estilos de los elementos, en determinado lapso de tiempo y con ciertas características. Para que una transición se ejecute es necesario disparar un evento (hover, onclick, focus, etc.)



**Transition-property:**

Determina la propiedad en la cual se aplicará la transición.

Sintaxis:

• transition-property: propiedad (background, color, opacity, height, width, etc.)

Ejemplo:

.caja {

*background*: red;

*transition-property*: background;

}

.caja:hover {

*background*: slateblue;

}

**Transition-duration:**

Define la duración de la transición. **Ejemplo**:

.caja {

*background*: red;

*transition-duration*: 1.0s;

}

**Transition-delay:**

Determina la cantidad de tiempo que va a transcurrir antes de que se ejecute la transición.

.caja {

*background*: red;

*transition-delay*: 2.5s;

}

**Transition-timing-function:**

Curva del tiempo en la que va a ejecutarse la transición. (Cambios en la aceleración y desaceleración.) Cabe resaltar que independientemente de la propiedad que elijamos, todas se terminarán de ejecutar en el mismo tiempo exacto.

**Valores**:

|  |  |
| --- | --- |
| **linear** | Velocidad constante |
| **ease** | (Por defecto) Leve retraso al final |
| **ease-in** | Retraso al inicio |
| **ease-out** | Retraso al final |
| **ease-in-out** | Retraso al inicio y al final |
| **steps int,** | start | end |
| **initial -** | initial - |
| **inherit** | Hereda la propiedad de la caja contenedora |

**Asignar valores simultáneamente:**

.caja {

*transition-property*: left, background, color;

*transition-duration*: 1s, 2.5s, 5s;

*transition-delay*: 0.3s, 0.5s, 0.7s;

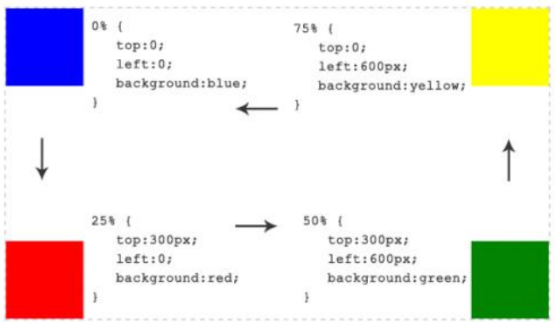
*transition-timing-function*: linear, ease-in, ease-out;

}

----------------------------------------------------------------------------------------

**ANIMATIONS EN CSS**

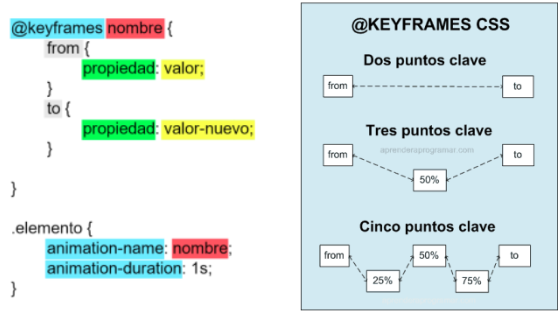
Las animaciones permiten realizar efectos visuales en base a una secuencia de estilos.



Keyframe = Punto clave

•Requiere de la propiedad **@keyframes.**

Para declarar los Keyframe de la secuencia, se trabaja con from / to (2 Keyframe) o con porcentajes (múltiples Keyframe).



**From / to:**

.caja {

*animation-name*: desplazarse;

*animation-duration*: 1s;

    }

    @keyframes *desplazarse* {

    from {

*position*: relative;

*left*: 0;

    }

    to {

*left*: 80%;

    }

}

**Porcentajes:**

.caja {

*animation-name*: desplazarse;

*animation-duration*: 1s;

}

@keyframes *desplazarse* {

    0% {

*position*: relative;

*left*: 0;

    }

    50% {

*left*: 30%;

    }

    100% {

*left*: 80%;

    }

}

**Animation-timing-function:**

Curva del tiempo en la que va a ejecutarse la animación. (Cambios en la aceleración y desaceleración).

|  |  |  |  |
| --- | --- | --- | --- |
| **linear** | Velocidad constante |  |  |
| **ease** | (Por defecto) Leve retraso al final |  |  |
| **ease-in** | Retraso al inicio |  |  |
| **ease-out** | Retraso al final |  |  |
| **ease-in-out** | Retraso al inicio y al final |  |  |
| **steps int,** | start | end |  |  |
| **initial -** | initial - |  |  |
| **inherit** | Hereda la propiedad de la caja contenedora |  |  |

**Animation-iteration-count:**

Determina la cantidad de veces que se va a repetir la animación.

1)

.caja {

*animation-iteration-count*: 3;

}

2)

.caja {

*animation-iteration-count*: infinite;

}

**Animation-direction:**

Cambia el orden de la secuencia de la animación. **Valores:**

• Normal (por defecto).

• Reverse.

• Alternate.

• Alternate-reverse.

.caja {

*animation-direction*: normal;

}

**Animation-fill-mode:**

Define cuál va a ser el modo final de la animación.

|  |  |
| --- | --- |
| **none -** | - |
| **backwards** | Se queda con los valores del PRIMER Keyframe. |
| **forward (recomendado)** | Se queda con los valores del ÚLTIMO Keyframe. |
| **both** | Si existe un delay, antes de iniciar la animación adopta los valores estáticos del primer Keyframe (background) |

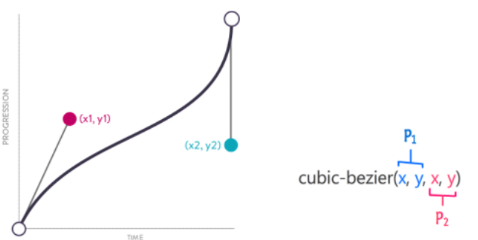
**Cubic Bezier:**

Función que se aplica en la propiedad animation-timing-function.

Permite modificar con máxima precisión la curva de tiempo de determinada animación. (aceleración y desaceleración)

• Se le otorgan 4 parámetros. (coordenadas en términos relativos)

• Recurso: <https://cubic-bezier.com>



Ejemplo:

.caja {

*animation-timing-function*:cubic-bezier(0,4, 0.2, 0.3, 1.2);

}

-------------------------------------------------------------------------------------------

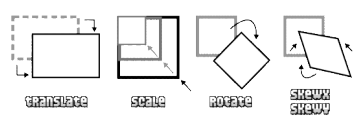
**TRANSFORM EN CSS**

Ejemplo:

.caja {

*transform*: translate(50px, -70%) rotate(30deg);

}



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**TRANSLATE**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Translate ()**

Desplaza a los elementos hacia cierta dirección.

Shorthand de TranslateX y TranslateY.

• Se recomienda utilizarlo por sobre las funciones top, left, right, bottom, etc. de position: relative, ya que posee una gran ventaja en cuanto a rendimiento y optimización.

Ejemplo:

.caja {

*transform*: translate(50px, -70%);

}

**TranslateX ()**

Desplaza a los elementos en el eje X (**horizontal**)

Ejemplo:

.caja {

*transform*: translateX(50px);

}

100% = width del elemento.

**TranslateY ()**

Desplaza a los elementos en el eje Y (**vertical**)

Ejemplo:

.caja {

*transform*: translateY(-70%);

}

100% = height del elemento.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**SCALE**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Scale ()**

Shorthand de ScaleX y ScaleY.

Estira determinado elemento (a través de vectores)

• Se le otorga un valor proporcional

Ejemplo:

.caja {

*transform*: scale(1.8, 2.5);

}

**ScaleX ()**

Estira determinado elemento en el eje X (**horizontal**)

• Se le otorga un valor proporcional

Ejemplo:

.caja {

*transform*: scaleX(1.5);

}

**ScaleY ()**

Estira determinado elemento en el eje Y (**vertical**)

• Se le otorga un valor proporcional

Ejemplo:

.caja {

*transform*: scaleY(2);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**SKEW**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Skew ()**

Deforma determinado elemento.

• Se le otorga un valor en grados sexagesimales (deg), centesimales (grad) o radianes (rad).

Ejemplos:

1)

.caja {

*transform*: skew(40deg);

}

2)

.caja {

*transform*: skew(-250deg);

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**ROTATE**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Rotate ()**

Rota determinado elemento.

• Se le otorga un valor en grados sexagesimales (deg), centesimales (grad) o radianes (rad).

Ejemplos:

.caja {

*transform*: rotate(30deg);

}

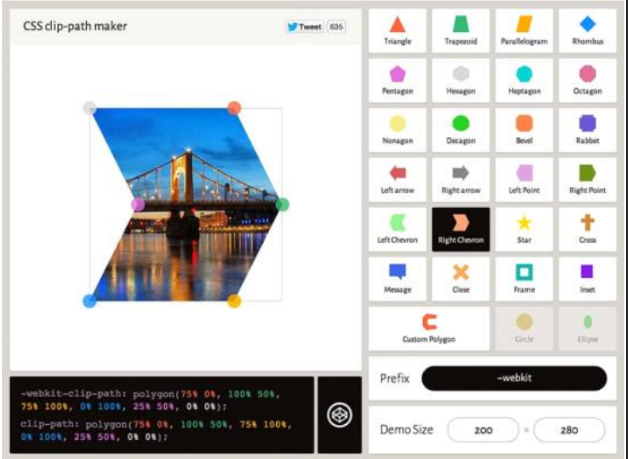
-------------------------------------------------------------------------------------------

**CLIP PATH EN CSS**

**Clip-path ()**

Permite recortar un elemento a diversas formas geométricas.

• Recurso: <https://bennettfeely.com/clippy/>



Ejemplo:

.caja {

*clip-path*: polygon(50% 0%, 0% 100%, 100% 100%);

}

-------------------------------------------------------------------------------------------

**BACKGROUND EN CSS**

**Background-color:**

Define el color de fondo.

.bg {

*background-color*: slateblue;

}

**Background-img:**

Coloca una imagen de fondo.

.bg {

*background-img: url(https://url-imagen);*

}

**Background-size:**

Determina el tamaño del fondo.

|  |  |
| --- | --- |
| **contain** | La imagen se ajusta a su resolución real, pero deja un espacio en el contenedor si es necesario. |
| **cover** | La imagen se ajusta al contenedor sin perder la resolución. |
| **none** | La imagen va a mantener su tamaño exacto sin importar el tamaño del contenedor. No se va a agrandar ni achicar. |
| **scale-down** | Se queda con la mejor propiedad dependiendo de la situación. |

.bg {

*background-size: cover;*

}

**Background-repeat:**

Determina si la imagen de fondo se va a repetir o no.

En el caso de que la imagen no se repita y quede un espacio vacío, este se rellenará con el color de fondo definido.

Valores: • repeat / • repeat-x / • repeat-y / • no-repeat

.bg {

*background-repeat*: no-repeat;

}

**Background-clip:**

Determina a partir de qué punto se va a mostrar la imagen de fondo. (Aplicando un recorte).

Valores: • content-box / • padding-box / • border-box

.bg {

*background-clip*: border-box;

}

**Background-origin:**

Determina desde qué punto se crea la imagen de fondo, **SIN RECORTARSE**.

Valores: • content-box / • padding-box / • border-box

.bg {

*background-origin*: border-box;

}

**Background-position:**

Determina la posición de la imagen de fondo.

Valores: • left / • right / • top / • bottom / • center

.bg {

*background-position*: left top;

}

**Background-attachment:**

Valores: • scroll (por defecto) / • fixed (imagen fija) / • inherit (heredado del contenedor).

.bg {

*background-attachment*: fixed;

}

-------------------------------------------------------------------------------------------

**VARIABLES EN CSS**

Una variable es un espacio que se almacena en memoria y guarda un valor.

Existen dos tipos de variables: globales y locales.

• Las **GLOBALES** son aquellas que pueden invocarse desde cualquier elemento.

Las **LOCALES** son aquellas que se declaran solo para ser invocadas por determinado tipo de elemento, clase, id, etc.

**Declarar una variable en CSS:**

**Global:**

:root {

    --color-rojo: #f43;

}

.caja {

*background*: var(--color-rojo);

}

**Local:**

div {

    --color-rojo: #f43;

}

div div {

*background*: var(--color-rojo);

}

**----------------------------------------------------------------------------------------**

**FILTROS EN CSS**

Propiedad que permite aplicar y ajustar efectos gráficos en los elementos. (Especialmente imágenes).



|  |  |  |
| --- | --- | --- |
| Filtros | Unit | Funcion |
| **none** | px | Por defecto |
| **blur** | px | Desenfoque gaussiano |
| **Brightness ()** | number or % | Brillo |
| **Contrast ()** | number or % | Contraste |
| **drop-shadow ()** | px - px - px - color | Sombra (img transparentes) |
| **Grayscale ()** | number or % | Escala de grises |
| **hue-rotate ()** | deg - rad - grad | Rotar la gama de colores |
| **Invert ()** | number or % | Invertir la gama de colores |
| **Opacity ()** | number or % | Opacidad |
| **Satúrate ()** | number or % | Saturación |
| **sepia ()** | number or % | Efecto sepia |
| **url ()** | ("filters.svg#filter.id") | Personalizado |

.img {

*filter*:

    brightness (2.3)

    grayscale (40%)

    contrast (1.5)

    drop-shadow (10px 10px 5px #000);

}

-------------------------------------------------------------------------------------------

**OTRAS PROPIEDADES EN CSS**

**Direction:**

Ordena el texto de determinada forma.

|  |  |
| --- | --- |
| **ltr** | De izquierda a derecha |
| **rtl** | De derecha a izquierda |
| **initial** | Por defecto |
| **inherit** | Valor heredado |

**Letter-spacing:**

Separación entre letras.

.caja {

*letter-spacing*: 1px;

}

**Scroll-behavior:**

Sensibilidad del scroll al redirigirte de una parte de la página a otra.

Valores:

• Smooth (progresivo).

• Auto (brusco).

**User-selected:**

Definir si determinado texto va a poder ser seleccionado, o no.

Su función es principalmente que cierto texto no pueda ser copiado.

Valores:

• None (no se puede seleccionar)

• Auto (sí se puede seleccionar)

**Text-shadow:**

Le otorga sombra a determinado texto.

Ejemplo:

.caja {

*text-shadow*: (2px 2px 1px #000);

}

**Text-decoration:**

Le otorga determinado estilo al texto.

Valores:

• underline / • dotted / • wavy / • overline / • none

Ejemplo:

.caja {

*text-decoration*: underline;

}

**Más selectores:**

Recurso: <https://www.w3schools.com/cssref/css_selectors.php>