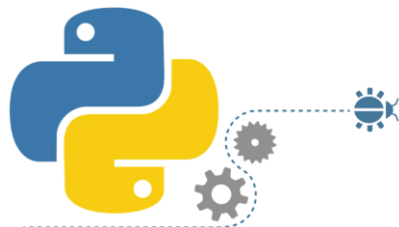


Curso FullStack Python

Codo a Codo 4.0



CSS

Parte 4

CSS



Selectores avanzados

Permiten ir más allá de la selección básica de los elementos. Utilizan “combinadores”, signos gráficos que establecen la relación entre los elementos y permiten hacer una selección **específica**. Tenemos varios métodos para seleccionar elementos dependiendo de la estructura del documento HTML:

Agrupación de selectores

Utilizaremos la , (coma) cuando varios elementos comparten una serie de declaraciones iguales. En lugar de crear varias reglas iguales en las que sólo cambia el selector, se crea una única regla con todos los selectores necesarios para apuntar a los distintos elementos. Esto ahorra tiempo de descarga:

```
p, a, div { /*Reglas CSS*/ }
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_grouping

Selectores descendientes

El *espacio en blanco* se utiliza para apuntar a elementos contenidos dentro de otro en el DOM del documento. Por ejemplo: seleccionar a todos los elementos **p** contenidos dentro de **div** sin importar la profundidad o los descendientes interpuestos entre **p** y **div**.

```
div p { /*Reglas CSS*/ }
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_element



Selector de hijos directos

El `>` (*mayor que*) se utiliza cuando queremos seleccionar a aquellos elementos que sean **hijos directos del contenedor** padre, descartando nietos y sucesivos. Por ejemplo: sólo se aplica a los elementos **a** contenidos directamente en **span** (**a** es hijo directo de **span**)

```
span > a { /*Reglas CSS*/ }
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_gt

Selector hermano adyacente

El signo `+` (*mas*) permite aplicar estilos a elementos que siguen a otros, es decir que está directamente después de otro elemento específico. Los elementos hermanos deben tener el mismo elemento padre y "adyacente" significa "inmediatamente siguiente". No puede haber ningún otro hermano que los separe o se interponga entre ellos.

```
div + p { /*Reglas CSS*/ }
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_pluss

Selector general de hermanos

El signo `~` (*virgulilla o tilde de la ñ*) selecciona todos los elementos que son hermanos de un elemento especificado, sin la necesidad de que sean adyacentes. Por ejemplo: seleccionar todos los elementos `<p>` que son hermanos de los elementos `<div>`:

```
div ~ p { /*Reglas CSS*/ }
```

https://www.w3schools.com/css/tryit.asp?filename=trycss_sel_element_tilde

Selector universal

El * (*asterísco*) representa a cualquier elemento del DOM. Al ser utilizado como selector CSS aplicará a cualquier elemento contenido en el documento.

https://www.w3schools.com/css/tryit.asp?filename=trycss_syntax_universal

El selector universal no solamente permite seleccionar todos los elementos de un documento HTML, sino también aquellos que pertenezcan a un id o clase específica. En el ejemplo se observa cómo se seleccionan todos los elementos contenidos en un **div** cuyo id es '**menu**'.

```
div#menu * { /*Reglas CSS*/ }
```

Para ampliar:

https://www.w3.org/wiki/CSS/_Selectores_CSS

<https://lenguajecss.com/css/selectores/selectores-avanzados/>

https://www.w3schools.com/cssref/css_selectors.asp

Pseudoclases

Una pseudoclase es un selector que marca los elementos que están en un estado específico o tienen un comportamiento determinado. Todas las pseudoclases son una palabra precedida por dos puntos y todas se comportan del mismo modo. Seleccionan un fragmento del documento que está en un estado determinado y se comportan como si se hubiera añadido una clase a su HTML.

:first-child:

Se utiliza para representar al primer elemento entre un grupo de elementos hermanos dentro de un contenedor, es decir “el primer hijo de su padre”.

```
<div>
  <p> Párrafo 1 </p>
  <p> Párrafo 2 </p>
  <p> Párrafo 3 </p>
</div>
```

HTML

```
p:first-child {
  color: red;
}
```

CSS

Párrafo 1
Párrafo 2
Párrafo 3

https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_firstchild

:last-child:

Se utiliza para representar al último elemento entre un grupo de elementos hermanos dentro de un contenedor, es decir “el último hijo de su padre”.

```
<div>
```

```
  <p> Párrafo 1 </p>
```

```
  <p> Párrafo 2 </p>
```

```
  <p> Párrafo 3 </p>
```

```
</div>
```

HTML

```
p:last-child {  
  color: blue;  
}
```

CSS

Párrafo 1

Párrafo 2

Párrafo 3

https://www.w3schools.com/cssref/tryit.asp?filename=trycss3_last-child

:nth-child(n):

El selector coincide con cada elemento que es el n -ésimo hijo, independientemente del tipo, de su padre. n puede ser un número, una palabra clave o una fórmula.

```
<div>
```

```
  <p> Párrafo 1 </p>
```

```
  <p> Párrafo 2 </p>
```

```
  <p> Párrafo 3 </p>
```

```
</div>
```

HTML

```
p:nth-child(2) {  
  background: cyan;  
}
```

CSS

Párrafo 1

Párrafo 2

Párrafo 3

https://www.w3schools.com/cssref/tryit.asp?filename=trycss3_nth-child

:nth-child(n): Otros ejemplos

Tomemos como ejemplo la siguiente lista:

```
<ol>
```

```
<li> Item 1</li>
```

```
<li> Item 2</li>
```

```
<li> Item 3</li>
```

```
<li> Item 4</li>
```

```
<li> Item 5</li>
```

```
<li> Item 6</li>
```

```
<li> Item 7</li>
```

```
<li> Item 8</li>
```

```
<li> Item 9</li>
```

```
</ol>
```

HTML

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

CSS

```
li:nth-child(3n) {  
  background:  
  lightskyblue;  
}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

Selecciona a los
elementos 3, 6, 9

CSS

```
li:nth-child(2n) {  
  background:  
  lightgreen;  
}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

Selecciona a los
elementos 2, 4, 6... etc

:nth-child(n): Otros ejemplos

Tomemos como ejemplo la siguiente lista:

```
<ol>
```

```
<li> Item 1</li>
```

```
<li> Item 2</li>
```

```
<li> Item 3</li>
```

```
<li> Item 4</li>
```

```
<li> Item 5</li>
```

```
<li> Item 6</li>
```

```
<li> Item 7</li>
```

```
<li> Item 8</li>
```

```
<li> Item 9</li>
```

```
</ol>
```

HTML

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

CSS

```
li:nth-child(3n+4) {  
  background:  
  lightcoral;  
}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

CSS

```
li:nth-child(even) {  
  background: lightcoral;  
}  
li:nth-child(odd) {  
  background: lightgreen;  
}
```

1. Item 1
2. Item 2
3. Item 3
4. Item 4
5. Item 5
6. Item 6
7. Item 7
8. Item 8
9. Item 9

Even: pares.

Odd: impares



Para el tema pseudoclasas ver
ejemplo pseudoclasas-1 (.html y .css)

Selecciona, de a 3 elementos,
a partir del 4to elemento

Pseudoclasas para hipervínculos

Se aplican a las etiquetas `<a>`, que pueden tener cuatro estados:

- **:link** se refiere a un enlace que todavía no ha sido visitado.
- **:hover** se refiere a un elemento sobre el que se coloca el puntero del mouse.
- **:visited** se refiere a un enlace que ya ha sido visitado.
- **:active** se refiere a cualquier elemento que ha sido activado por el usuario.

```
<a href="https://google.com" target="_blank">Ir a Google</a>
```

HTML

```
a:link {color: red;}  
a:hover {background-color: yellow;}  
a:visited {color: blue;}  
a:active {background-color: green; color: white;}
```

CSS

Contacto

Contacto

Contacto

Contacto



Ver ejemplo pseudoclasas-a
(.html y .css)

Para seguir investigando:

https://www.w3schools.com/css/css_pseudo_classes.asp

Pseudoelementos

Se utilizan para darle estilos a partes específicas de un elemento. Están precedida por cuatro puntos (:):

::first-letter:

Se utiliza para darle estilo a la primer letra de un texto. En este caso afectamos al párrafo:

```
<p>Párrafo con la primera letra de otro color</p>
```

HTML

Párrafo con la pri

```
p::first-letter{color:blue;}
```

CSS

https://www.w3schools.com/cssref/sel_firstletter.asp

::first-line:

Se utiliza para darle estilo a la primer línea de un párrafo:

```
p::first-line{background-color: lightgreen;}
```

CSS

Lorem ipsum dolor sit amet consectetur adipisicing elit.
Deleniti quaerat asperiores vitae aspernatur ut incidunt
dolores tempora saepe harum at, ullam laudantium

*Siempre afectará a la primer línea,
independientemente del ancho del viewport*

https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_firstline

::selection:

Agrega estilos a una parte del documento que ha sido resaltada por el usuario:

```
p::selection{background-color: lightsalmon;}
```

CSS

Lorem ipsum dolor sit amet consectetur
harum at, ullam laudantium consectetur
molestias eius, magnam explicabo hic a

https://www.w3schools.com/cssref/tryit.asp?filename=trycss3_selection

::before y ::after:

::before agrega contenido **antes** del contenido, mientras que **::after** lo añade **después** del contenido:

```
p::before{ content: "🌟"; }  
p::after{ content: "🐼"; }
```

CSS

🌟 Texto de ejemplo 🐼

https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_after
https://www.w3schools.com/cssref/tryit.asp?filename=trycss_sel_before



[Ver ejemplo pseudoelementos-1 \(.html y .css\)](#)

Transformaciones

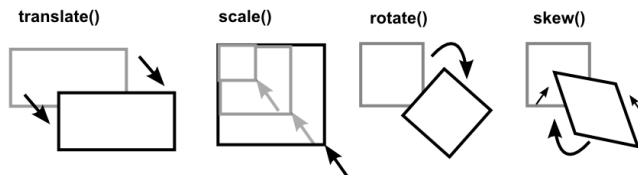
Las transformaciones CSS le permiten mover, rotar, escalar y sesgar elementos, es decir, todo tipo de efectos visuales, incluido 2D y 3D. Las propiedades principales para realizar transformaciones son las siguientes:

Propiedades	Formato	Significado
<code>transform</code>	<i>función1, función2, ...</i>	Aplica una o varias funciones de transformación sobre un elemento.
<code>transform-origin</code>	<code>POSX</code> <code>POSY</code> <code>POZY</code>	Cambia el punto de origen del elemento en una transformación.
<code>transform-style</code>	<code>flat</code> <code>preserve-3d</code>	Modifica el tratamiento de los elementos hijos.

Con la propiedad **transform** podemos indicar una o varias transformaciones para realizar sobre un elemento, ya sean 2D (sobre dos ejes) o 3D (sobre tres ejes).

Funciones 2D

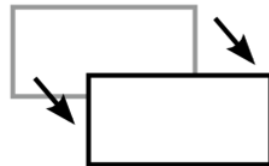
Existen múltiples propiedades CSS que ofrecen diferentes funcionalidades de transformación en dos dimensiones.



Traslaciones (translate)

Las **funciones de translación** son aquellas que realizan una transformación en la que **mueven** un elemento de un lugar a otro. Si especificamos un valor positivo en el eje X (*horizontal*), lo moveremos hacia la derecha, y si especificamos un valor negativo, lo moveremos hacia la izquierda. Lo mismo con el eje Y (*vertical*).

translate()



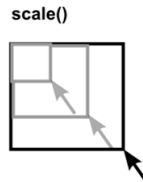
Funciones	Significado
<code>translateX(x)</code>	Traslada el elemento una distancia de <code>SIZE</code> <code>x</code> horizontalmente.
<code>translateY(y)</code>	Traslada el elemento una distancia de <code>SIZE</code> <code>y</code> verticalmente.
<code>translate(x, y)</code>	Propiedad de atajo de las dos anteriores.

Por ejemplo, **transform: translate(20px, -30px)** traslada el elemento 20 píxeles a la derecha y 30 píxeles hacia arriba, que es equivalente a utilizar **transform: translateX(20px) translateY(-30px)**.

https://www.w3schools.com/css/tryit.asp?filename=trycss3_transform_translate

Escalado (scale)

Las **funciones de escalado** realizan una transformación en la que aumentan o reducen el tamaño de un elemento, basándose en el parámetro indicado, que no es más que un factor de escala:



Funciones	Significado
<code>scaleX(fx)</code>	Reescala el elemento a un nuevo tamaño con un factor <code>NUMBER</code> fx horizontal.
<code>scaleY(fy)</code>	Reescala el elemento a un nuevo tamaño con un factor <code>NUMBER</code> fy vertical.
<code>scale(fx, fy)</code>	Propiedad de atajo de las dos anteriores.

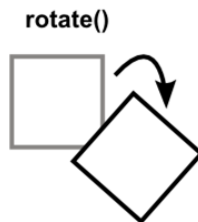
En este ejemplo, **transform: scale(2, 2)** realiza una transformación de escalado del elemento, ampliándolo al doble de su tamaño original. Si utilizamos **scale()** con dos parámetros iguales, estamos manteniendo la proporción del elemento, pero si utilizamos diferentes valores, acabaría deformándose.

https://www.w3schools.com/css/tryit.asp?filename=trycss3_transform_scale

Rotaciones (rotate)

Las funciones de rotación simplemente giran el elemento el número de grados indicado.

Funciones	Significado
<code>rotateX(xdeg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>xdeg</i> grados sólo para el eje horizontal X.
<code>rotateY(ydeg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>ydeg</i> grados sólo para el eje vertical Y.
<code>rotate(deg)</code>	Establece una rotación 2D en <code>DIRECTION</code> <i>deg</i> grados sobre si mismo.

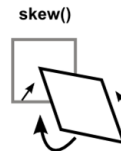


Con **transform: rotate(5deg)** realizamos una rotación de 5 grados del elemento sobre si mismo. Utilizando **rotateX()** y **rotateY()** podemos hacer lo mismo respecto al eje X o el eje Y respectivamente.

https://www.w3schools.com/css/tryit.asp?filename=trycss3_transform_rotate

Deformaciones (skew)

Por último, las **funciones de deformación** establecen un ángulo para torcer, tumbar o inclinar un elemento en 2D.



Funciones	Significado
<code>skewX(xdeg)</code>	Establece un ángulo de DIRECTION <i>xdeg</i> para una deformación 2D respecto al eje X
<code>skewY(ydeg)</code>	Establece un ángulo de DIRECTION <i>ydeg</i> para una deformación 2D respecto al eje Y

Aunque la función **skew()** existe, no debería ser utilizada, ya que está marcada como obsoleta y serán retiradas de los navegadores en el futuro. En su lugar deberían utilizarse **skewX()** o **skewY()**.

https://www.w3schools.com/css/tryit.asp?filename=trycss3_transform_skewx

Para seguir investigando: https://www.w3schools.com/css/css3_2dtransforms.asp



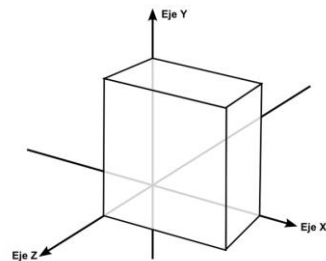
[Ver ejemplo transformaciones.html](#)

Funciones 3D

A las funciones anteriores, también podemos añadir las funciones equivalentes de CSS para hacer uso del eje Z (tres dimensiones o 3D). Basta con utilizar el eje Z o las funciones específicas de 3D para poner estas animaciones en práctica.

Las propiedades de transformación que completarían la colección de transformaciones 2D que vimos anteriormente, son las siguientes:

Funciones	Significado
<code>translateZ(z)</code>	Traslada el elemento una distancia de SIZE <code>z</code> en el eje de profundidad.
<code>translate3d(x, y, z)</code>	Establece una translación 3D, donde aplica los parámetros de SIZE a cada eje.
<code>scaleZ(fz)</code>	Reescala el elemento a un nuevo tamaño con factor NUMBER <code>fz</code> de profundidad.
<code>scale3d(fx, fy, fz)</code>	Establece un escalado 3D, donde aplica los factores a cada eje.
<code>rotateZ(zdeg)</code>	Establece una rotación 2D en DIRECTION <code>zdeg</code> grados sólo para el eje de profundidad Z.
<code>rotate3d(x, y, z, deg)</code>	Establece una rotación 3D, aplicando un vector <code>[x,y,z]</code> y el ángulo en DIRECTION <code>deg</code> .
<code>perspective(n)</code>	Establece una perspectiva 3D de SIZE <code>n</code>
<code>matrix3d(n, n, ...)</code>	Establece una matriz de transformación 3D (<i>16 valores</i>)



Para ampliar: https://www.w3schools.com/css/css3_3dtransforms.asp

Transformaciones múltiples

Al establecer varias propiedades transform en el mismo elemento con diferentes funciones de transformación, la segunda propiedad de transformación sobrescribirá a la anterior, como lo haría cualquier propiedad de CSS:

```
div {  
  transform: rotate(5deg);  
  transform: scale(2,2);    /* Sobrescribe la anterior */  
}
```

CSS

Para evitar este comportamiento, una forma sencilla se basa en emplear múltiples transformaciones separándolas mediante un espacio. En el siguiente ejemplo, aplicamos una función de rotación, una función de escalado y una función de traslación de forma simultánea:

```
div {  
  transform: rotate(5deg) scale(2,2) translate(20px, 40px);  
}
```

CSS

Transiciones


Las transiciones CSS le permiten cambiar los valores de una propiedad, durante un período determinado. Se basan en un principio muy básico: conseguir un **cambio de estilo** con un efecto suavizado entre un estado inicial y un estado final.

Para crear un efecto de transición, debemos especificar dos cosas:

- la propiedad CSS a la que desea agregar un efecto (*¿qué propiedad modifico?*)
- la duración del efecto (*¿durante cuánto tiempo?*)

Las propiedades relacionadas que existen son las siguientes:

Propiedades	Valor
<code>transition-property</code>	<code>all</code> <code>none</code> <u>propiedad css</u>
<code>transition-duration</code>	<code>0</code> <code>TIME</code>
<code>transition-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier(<u>A</u>, <u>B</u>, <u>C</u>, <u>D</u>)</code>
<code>transition-delay</code>	<code>0</code> <code>TIME</code>



transition-property: Se utiliza para especificar la **propiedad a la que afectará la transición**. Podemos especificar la propiedad concreta (***width** o **color**, por ejemplo*) o simplemente especificar **all** para que se aplique a todos los elementos con los que se encuentre. Por otro lado, **none** hace que no se aplique ninguna transición.

transition-duration: Permite establecer la **duración de la transición**, desde su inicio hasta su finalización. Se recomienda siempre comenzar con valores cortos, para que las transiciones sean rápidas y elegantes.

Si establecemos una duración demasiado grande, el navegador realizará la transición con detenciones intermitentes, lo que hará que la transición vaya a golpes. Además, transiciones muy largas pueden resultar molestas a muchos usuarios.

transition-timing-function: indica el **ritmo de la transición** que queremos conseguir. Cuando estamos aprendiendo CSS, recomiendo utilizar **linear**, que realiza una transición a un ritmo constante. Sin embargo, podemos utilizar otros valores para conseguir que el ritmo sea diferente al inicio y/o al final de la transición.

(continúa....)



Los valores que puede tomar la propiedad son los siguientes:

Valor	Inicio	Transcurso	Final	Equivalente en cubic-bezier
ease	Lento	Rápido	Lento	(0.25, 0.1, 0.25, 1)
linear	Normal	Normal	Normal	(0, 0, 1, 1)
ease-in	Lento	Normal	Normal	(0.42, 0, 1, 1)
ease-out	Normal	Normal	Lento	(0, 0, 0.58, 1)
ease-in-out	Lento	Normal	Lento	(0.42, 0, 0.58, 1)
cubic-bezier(<u>A</u> , <u>B</u> , <u>C</u> , <u>D</u>)	-	-	-	Transición personalizada

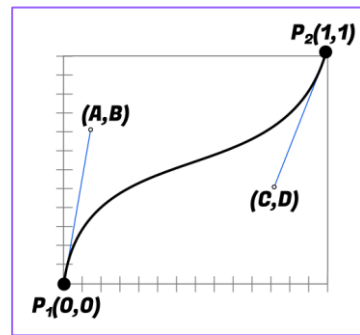
Una función de tiempo **linear** siempre es constante, mientras que **ease** comienza suavemente, continua de forma más rápida y termina suavemente de nuevo. **Ease-in** y **ease-out** son variaciones que van más lento al principio o al final, y **ease-in-out** una mezcla de las dos.

Encontramos también la función **Cubic-Bezier()** que nos permite configurar con más detalle la transición.

La función de tiempo Cubic-Bezier()

Es una función personalizada, donde podemos darle unos valores concretos dependiendo de la velocidad que queramos que tenga la transición. En la última columna de la tabla anterior podemos ver los valores equivalentes a cada una de las palabras clave mencionadas. En principio, el formato de la función es **cubic-bezier(A, B, C, D)**, donde:

Parámetro	Valor	Descripción	Pertenece a
A	X_1	Eje X del primer punto que orienta la curva bezier.	P_1
B	Y_1	Eje Y del primer punto que orienta la curva bezier.	P_1
C	X_2	Eje X del segundo punto que orienta la curva bezier.	P_2
D	Y_2	Eje Y del segundo punto que orienta la curva bezier.	P_2



Para simular el efecto: <https://cubic-bezier.com/#.17,.67,.83,.67>

transition-delay: Nos ofrece la posibilidad de **retrasar el inicio de la transición** los segundos especificados.



Ver ejemplos transiciones-1, transiciones-2 (.html y .css)

Atajo: Transiciones

Como siempre, podemos resumir todas estas operaciones en una propiedad de atajo denominada **transition**. Los valores del ejemplo superior, se podrían escribir como se puede ver a continuación (*si no necesitas algún valor, se puede omitir*):

```
div {  
  /* transition: <property> <duration> <timing-function> <delay> */  
  transition: all 0.2s ease-in;  
}
```

CSS

Fuente: <https://lenguajecss.com/css/animaciones/transiciones/>

Para seguir investigando:

https://www.w3schools.com/css/css3_transitions.asp
(se recomiendan especialmente los últimos ejemplos que son más completos)





Animaciones

Una animación permite que un elemento cambie gradualmente de un estilo a otro. Podemos cambiar tantas propiedades CSS como deseemos, tantas veces como deseemos.

Las animaciones amplían el concepto de transiciones convirtiéndolo en algo mucho más flexible y potente, partiendo del mismo concepto de realizar cambios en ciertos estados inicial y final pero incorporando más estados, pudiendo realizar cambios desde un estado inicial, a un estado posterior, a otro estado posterior, y así sucesivamente. Además, esto será posible de forma automática, **sin que el usuario tenga que realizar una acción concreta.**

Para utilizar la animación CSS, primero debemos especificar algunos fotogramas clave (*@keyframes*) para la animación, que contendrán los estilos que tendrá el elemento en determinados momentos. Además tendremos que utilizar las propiedades de las animaciones, que definen el comportamiento de la misma.



Propiedades de animación CSS

Para definir dicho comportamiento necesitamos conocer las siguientes propiedades, que son una ampliación de las transiciones CSS:

Propiedades	Valor
<code>animation-name</code>	<code>none</code> <i>nombre</i>
<code>animation-duration</code>	<code>0</code> <code>TIME</code>
<code>animation-timing-function</code>	<code>ease</code> <code>linear</code> <code>ease-in</code> <code>ease-out</code> <code>ease-in-out</code> <code>cubic-bezier</code> (<i>A</i> , <i>B</i> , <i>C</i> , <i>D</i>)
<code>animation-delay</code>	<code>0</code> <code>TIME</code>
<code>animation-iteration-count</code>	<code>1</code> <code>infinite</code> <code>NUMBER</code>
<code>animation-direction</code>	<code>normal</code> <code>reverse</code> <code>alternate</code> <code>alternate-reverse</code>
<code>animation-fill-mode</code>	<code>none</code> <code>forwards</code> <code>backwards</code> <code>both</code>
<code>animation-play-state</code>	<code>running</code> <code>paused</code>

La propiedad **animation-name** permite especificar el nombre del fotograma a utilizar, mientras que las propiedades **animation-duration**, **animation-timing-function** y **animation-delay** funcionan exactamente igual que en *transiciones*.

La propiedad **animation-iteration-count** permite indicar el número de veces que se repite la animación, pudiendo establecer un número concreto de repeticiones o indicando **infinite** para que se repita continuamente. Por otra parte, especificando un valor en **animation-direction** conseguiremos indicar el orden en el que se reproducirán los fotogramas, pudiendo escoger un valor de los siguientes:

Valor	Significado
normal	Los fotogramas se reproducen desde el principio al final.
reverse	Los fotogramas se reproducen desde el final al principio.
alternate	En iteraciones par, de forma normal. Impares, a la inversa.
alternate-reverse	En iteraciones impares, de forma normal. Pares, normal.

Por defecto, cuando se termina una animación que se ha indicado que se reproduzca sólo una vez, la animación vuelve a su estado inicial (*primer fotograma*). Mediante la propiedad **animation-fill-mode** podemos indicar que debe mostrar la animación cuando ha finalizado y ya no se está reproduciendo; si mostrar el estado inicial (*backwards*), el estado final (*forwards*) o una combinación de ambas (*both*).

La propiedad **animation-play-state** nos permite establecer la animación a estado de reproducción (*running*) o pausarla (*paused*).

Atajo: Animaciones

Nuevamente, CSS ofrece la posibilidad de resumir todas estas propiedades en una sola, para hacer nuestras hojas de estilos más específicas. El orden de la propiedad de atajo sería el siguiente:

```
div {  
  /* animation: <name> <duration> <timing-function> <delay>  
    <iteration-count> <direction> <fill-mode> <play-state> */  
  animation: changeColor 5s linear 0.5s 4 normal forwards running;  
}
```

CSS

Consejo: Mucho cuidado al indicar los segundos en las propiedades de duración. Al ser una unidad diferente a las que solemos manejar (px, em, etc...) hay que especificar **siempre** la **s**, aunque sea un valor igual a 0.

Fotogramas (keyframes)

Para definir los fotogramas de una animación utilizaremos la regla **@keyframes**, la cual es muy sencilla de utilizar y se basa en el siguiente esquema:

```
@keyframes nombre {  
  selectorkeyframe {  
    propiedad: valor;  
    propiedad: valor;  
  }  
}
```

En primer lugar elegiremos un **nombre** para la animación (*el cual utilizamos en el apartado anterior, para hacer referencia a la animación, ya que podemos tener varias en una misma página*), mientras que podremos utilizar varios selectores para definir el transcurso de los fotogramas en la animación.

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: blue;  
  animation-name: cambiarColor;  
  animation-duration: 2s;  
  animation-delay: 1s;  
  /* animation: cambiarColor 2s 1s; */  
}
```

CSS

```
@keyframes cambiarColor {  
  from {background-color: red;}  
  to {background-color: yellow;}  
}
```

CSS



En este ejemplo nombrado **cambiarColor**, partimos de un primer fotograma en el que el elemento en cuestión será de color de fondo rojo. Si observamos el último fotograma, le ordenamos que termine con el color de fondo verde. Así pues, la regla **@keyframes** se inventará la animación intermedia para conseguir que el elemento cambie de color.

Los selectores **from** y **to** son realmente sinónimos de 0% y 100%. Al modificarlos podremos ir añadiendo nuevos fotogramas intermedios. Vamos a modificar el ejemplo anterior:

```
div {  
  width: 100px;  
  height: 100px;  
  background-color: blue;  
  animation-name: cambiarColor;  
  animation-duration: 2s;  
  animation-timing-function: ease;  
  animation-iteration-count: infinite;  
}
```

CSS

```
@keyframes cambiarColor {  
  0% {background: red; width: 200px;}  
    /* Primer fotograma */  
  50% {background: yellow; width: 400px;}  
    /* Segundo fotograma */  
  100% {background: green; width: 600px;}  
    /* Último fotograma */  
}
```

CSS



[Ver ejemplos animaciones-1, animaciones-2 y animaciones-3 \(.html y .css\)](#)

Encadenar animaciones

Es posible encadenar múltiples animaciones, separando con comas las animaciones individuales y estableciendo un tiempo de tardeo a cada animación posterior:

```
.animated {  
  animation:  
    moveRight 5s linear 0,      /* Comienza a los 0s */  
    lookUp 2.5s linear 5s,     /* Comienza a los 5s */  
    moveLeft 2s linear 7.5s,   /* Comienza a los 7.5s (5 + 2.5) */  
    disappear 2s linear 9.5s; /* Comienza a los 9.5s (2 + 7.5) */  
}
```

CSS

En este caso, lo que hemos hecho es aplicar varias animaciones a la vez, pero estableciendo un retardo (*cuarto parámetro*) que es la suma de la duración de las animaciones anteriores. De esta forma, encadenamos una animación con otra.

Para seguir investigando:

https://www.w3schools.com/css/css3_animations.asp

(se recomiendan especialmente los últimos ejemplos sobre movimiento de elementos)

Librería de animaciones Animate.css

- Podemos utilizar Animate.css para dar dinamismo a nuestro contenido. Link: <https://animate.style/>
- En pocos pasos se pueden agregar animaciones CSS3 a cualquier elemento con esta sencilla librería.
- En la creación de cualquier contenido web puede resultarnos interesante incorporar animaciones que nos ayuden a mejorar la experiencia del usuario durante la interacción con el contenido.
- Permite disponer de una gran variedad de animaciones CSS3 sin necesidad de crearlas nosotros mismos.
- Esta librería permite conseguir que el contenido sea más atractivo y dinámico.



[Ver ejemplo animate-css.html](#)

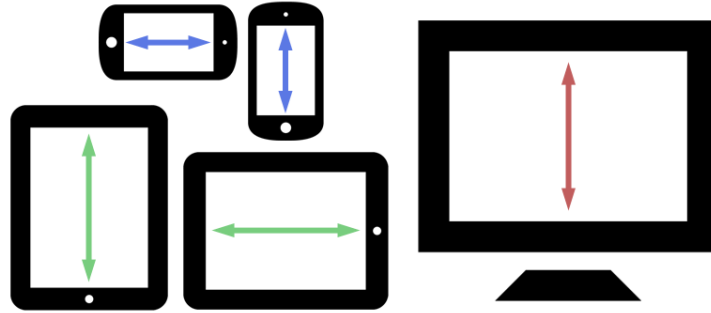
Para ampliar:

<https://blog.interactius.com/utilizando-animate-css-para-dar-dinamismo-a-nuestro-contenido-64d280d4d119>

<http://www.elpadawan.com/css/animatecss>

Diseño Web Responsivo (introducción)

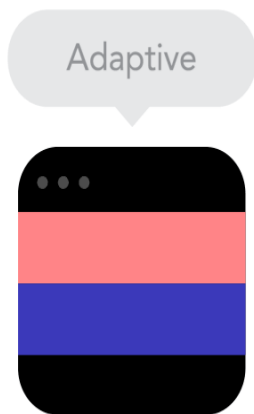
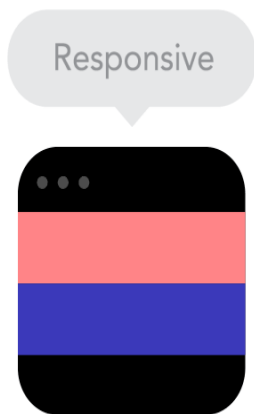
El diseño web responsivo se trata de usar HTML y CSS para cambiar el tamaño, ocultar, reducir o ampliar automáticamente un sitio web, para que se vea bien en todos los dispositivos (computadoras de escritorio, tabletas y teléfonos).



Para ampliar: https://www.w3schools.com/css/css_rwd_intro.asp

Diseño responsivo vs Diseño adaptativo

Un diseño responsive responde **en todo momento** a las dimensiones del dispositivo, mientras que un diseño adaptativo es aquel que se adapta, pero **no necesariamente responde en todo momento**, tiene cierto delay, estamos hablando casi de lo mismo.



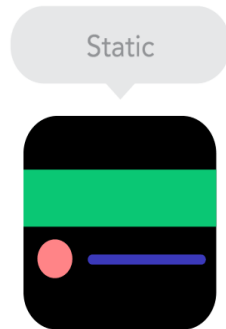
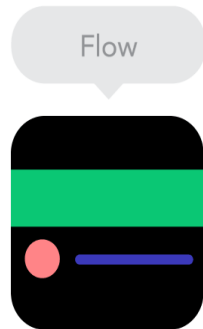
El diseño web responsive **adapta la estructura y los diferentes elementos** de cada página web a las dimensiones y características de cada aparato móvil. Por otro lado, el diseño web adaptativo es **menos flexible**, y se basa en el uso de tamaños y características pre-establecidas. Las diferencias entre ambos métodos no se encuentran solamente en el resultado final y la experiencia del usuario; también se encuentran en el proceso creativo y de diseño.

Fuente: 9 principios básicos del diseño Web responsive. Click [aquí](#)

Flujo (The Flow) vs Estático (Static)

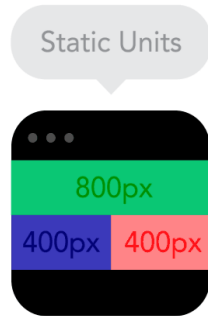
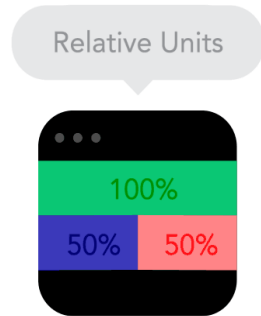
Cuando una pantalla se vuelve más pequeña, el contenido comienza a crecer verticalmente ocupando más espacio, y el contenido que se encuentra debajo va a ser desplazado hacia abajo, eso se llama **el flujo**.

Si es estático ese flujo de elementos no se deslaza, no se adapta al ancho del viewport y se pierde contenido o cierto contenido tapa a otro.



Unidades Relativas vs Unidades Absolutas

La densidad de píxeles de cada dispositivo puede variar, por eso necesitamos unidades que sean flexibles y funcionen sin importar el dispositivo. Ahí es donde las unidades relativas como los porcentajes son útiles. Entonces, hacer algo con un 50% de ancho significa que siempre ocupará la mitad de la pantalla (viewport, el tamaño de la ventana del navegador abierta), independientemente del dispositivo.



Valores Mínimos y Máximos

En un celular nos puede interesar que determinado contenido ocupe todo el ancho de la pantalla, pero al pasar a un televisor de alta definición podríamos cambiar de idea. Por ejemplo podríamos tener un **width:100%**, pero con un **max width: 1000px**.

Si la imagen la va a cargar en un celular no necesariamente tiene que tener gran calidad o cierto ancho, distinto si lo fuera a cargar en un dispositivo más grande Ultra HD. El alto no importa tanto en mobile, porque podemos scrollar, si importa el ancho.

Puntos de Control (Breakpoints)

Estos puntos de control permiten al diseño cambiar en determinados puntos, por ej, en un monitor podemos tener 3 columnas, pero sólo 1 en un celular (que es mas angosto). Estos puntos de control o de quiebre se crean con los **media queries**, que nos permitirán decir que si el mínimo del ancho de la pantalla en píxeles es tanto en vez de poner el contenido en tres columnas mostrámelos en una sola y creame tres filas.

Max width



No max width



With Breakpoints



Without Breakpoints



Puntos de Control (Breakpoints)

Hay muchos tipos de pantallas y dispositivos con diferentes alturas y anchos, por lo que es difícil crear un breakpoint para cada dispositivo. Para simplificar las cosas, se puede hacer lo siguiente:

```
<style>
  .example {
    padding: 20px;
    color: white;
  }
  /* Extra small devices (phones, 600px and down) */
  @media only screen and (max-width: 600px) {
    .example {background: red;}
  }
  /* Small devices (portrait tablets and large phones, 600px and up) */
  @media only screen and (min-width: 600px) {
    .example {background: green;}
  }
  /* Medium devices (landscape tablets, 768px and up) */
  @media only screen and (min-width: 768px) {
    .example {background: blue;}
  }

```

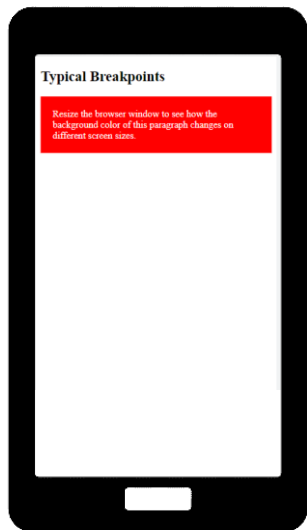
```
/* Large devices (laptops/desktops, 992px and up) */
@media only screen and (min-width: 992px) {
  .example {background: orange;}
}
/* Extra large devices (large laptops and desktops, 1200px and up) */
@media only screen and (min-width: 1200px) {
  .example {background: pink;}
}
</style>

```

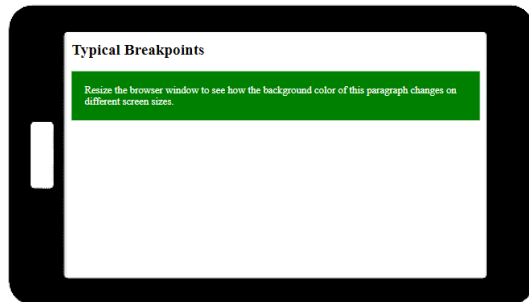
Puntos de corte (según ancho):

- Hasta 600 px: Fondo rojo
- Desde 600 px: Fondo verde
- Desde 768 px: Fondo azul
- Desde 992 px: Fondo naranja
- Desde 1200 px: Fondo rosa

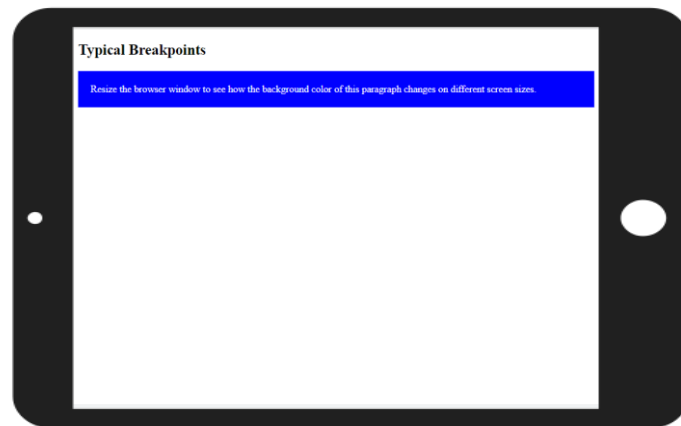
Puntos de Control (Breakpoints)



400px X 600px
Extra small devices
(phones, 600px and
down)

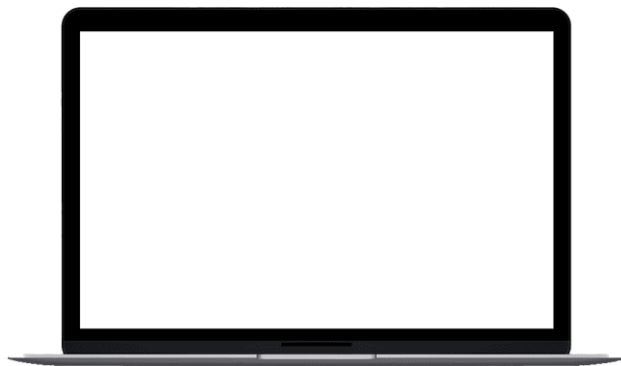


650px X 400px
Small devices (portrait
tablets and large phones,
600px and up)



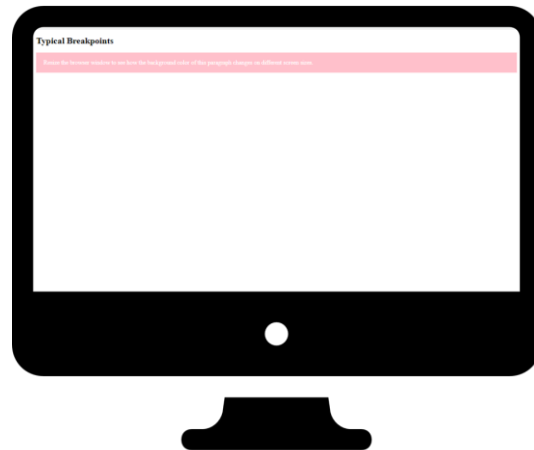
850px X 600px
Medium devices (landscape
tablets, 768px and up)

Puntos de Control (Breakpoints)



1000px X 800px

**Large devices (laptops/desktops,
992px and up)**



1300px X 800px

**Extra large devices (large laptops
and desktops, 1200px and up)**

Más información:

- Typical Device Breakpoints: click [aquí](#).
- How to use CSS breakpoints to create responsive designs: [click aquí](#).



[Ver ejemplo breakpoints.html](#)

Objetos anidados (Nested Objects)

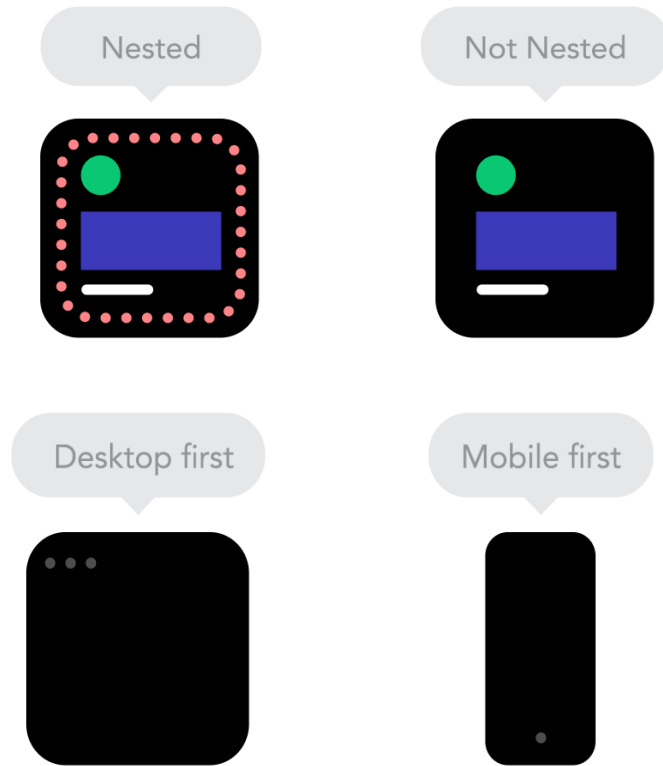
¿Recuerdan la **posición relativa**? Tener muchos objetos que dependan de otros puede ser difícil de controlar, sin embargo, agruparlos en contenedores nos puede simplificar las cosas.

¿Por qué usamos contenedores? Porque a la hora de pensar contenido responsive nos va a facilitar posicionar un grupo de elementos en otro lugar.

Mobile first vs Desktop first

- **Mobile first:** Primero nos enfocamos en dispositivos móviles y luego pensamos en otros.
- **Desktop first:** Primero nos enfocamos en dispositivos de escritorio, y luego pensamos en otros.

Si hacemos una medición de la navegación de los sitios Web los dispositivos mobile son por excelencia los dispositivos que tienden a acceder a los sitios Web, ya los dispositivos de escritorio tienden a bajar en las estadísticas en cuanto al % de acceso, la tendencia va más a mobile.



System Font vs WebFonts

- **Fuentes de la Web:** son descargadas por lo que, cuantas más haya, mas lento cargará el sitio.
- **Fuentes del Sistema:** más rápidas, pero si NO están en el cliente navegador del usuario se usa una por defecto.

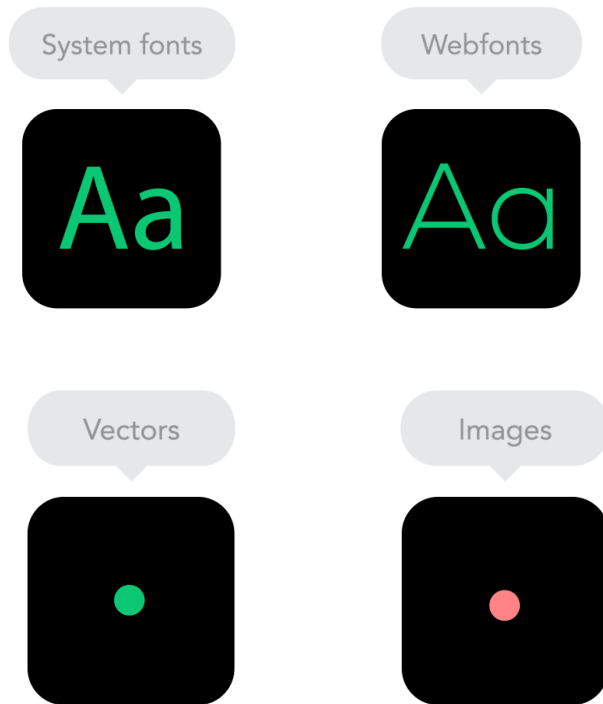
Cuando estamos trabajando con dispositivos móviles tenemos que tener en cuenta que todo se carga.

Bitmaps vs Vectors

- **Bitmaps:** JPG, PNG, GIF. Recomendadas para muchos detalles y efectos.
- **Vectors:** SVG (gráficos basados en vectores escalables), si voy a mostrar un ícono uso Icon Fonts, que son mas livianos, pero algunos exploradores viejos no los soportan.

Más sobre SVG: <https://desarrolloweb.com/articulos/que-es-svg.html>

Para saber si una tecnología es soportada por un navegador: <https://caniuse.com/>



Texto responsivo

El tamaño del texto se puede configurar con una unidad "vw", que es el "ancho de la ventana gráfica".

De esa forma, el tamaño del texto seguirá el tamaño de la ventana del navegador.

```
<h1 style="font-size:10vw">Hello World</h1>
```

Viewport es el tamaño de la ventana del navegador.

1vw = 1% del ancho de la ventana gráfica.

Si la ventana tiene 50 cm de ancho, 1 vw es 0,5 cm.

Diseño Web Responsivo: https://www.w3schools.com/html/html_responsive.asp

Practica:

https://www.w3schools.com/html/tryit.asp?filename=tryhtml_responsive_page

Para seguir investigando: https://www.w3schools.com/html/html_responsive.asp

Imágenes responsivas

Las imágenes responsivas son imágenes que se escalan bien para adaptarse a cualquier tamaño de navegador.

Si la propiedad CSS **width** se establece en 100%, la imagen responderá y se ampliará y reducirá.

Una imagen grande puede ser perfecta en una pantalla de computadora grande, pero inútil en un dispositivo pequeño. ¿Por qué cargar una imagen grande cuando tiene que reducirla de todos modos? Para reducir la carga, o por cualquier otro motivo, puede utilizar **media queries** para mostrar diferentes imágenes en diferentes dispositivos.

Imágenes responsivas: https://www.w3schools.com/css/css_rwd_images.asp

Display

Display es la propiedad más importante para controlar estructuras. Cada elemento tiene un valor de display por defecto, ya vimos los valores por defecto **block** e **inline** que los navegadores le dan a los elementos.

- **block**: un elemento block empieza en una nueva línea ya lo vimos en elementos como div, h1-h6, header, etc.
- **inline**: Un elemento inline puede contener algo de texto dentro de un párrafo sin interrumpir el flujo del párrafo.
- **none**: es utilizado para ocultar elementos sin eliminarlos, no deja un espacio donde el elemento se encontraba.
- **inline-block**: Los elementos inline-block fluyen con el texto y demás elementos como si fueran elementos en-línea y además respetan el ancho, el alto y los márgenes verticales.

Más información: https://www.w3schools.com/cssref/pr_class_display.asp

Cada etiqueta HTML tiene un tipo de representación visual en un navegador, lo que habitualmente se suele denominar el **tipo de caja**. En principio, se parte de dos tipos básicos: **inline** y **block**.

Valor	Denominación	Significado	Ejemplo
inline	Elemento en línea	El elemento se coloca en horizontal (un elemento a continuación del otro).	<code></code>
block	Elemento en bloque	El elemento se coloca en vertical (un elemento encima de otro).	<code><div></code>

Obsérvese que por defecto, todos los elementos `<div>` son elementos de bloque (*block*) y todos los elementos `` son elementos en línea (*inline*). Para entender esto fácilmente, veamos este HTML con 3 etiquetas `<div>` en la imagen de la derecha:

```
<div>Elemento 1</div>
<div>Elemento 2</div>
<div>Elemento 3</div>
```

A estas etiquetas HTML le vamos a aplicar el siguiente código CSS:

```
div {
  background: blue;
  color: white;
  margin: 1px;
}
```

Con esto observaremos que en nuestro navegador nos aparecen 3 cajas azules colocadas en vertical (*una debajo de otra*) que cubren todo el ancho disponible de la página. Esto ocurre porque la etiqueta `<div>` es un elemento en bloque, o lo que es lo mismo, que tiene un tipo de representación **block** por defecto. Cada etiqueta HTML tiene un tipo de representación concreta.

Sin embargo, este comportamiento de elementos puede cambiarse con la propiedad CSS **display**. Tan sencillo como añadir **display: inline** en el ejemplo anterior y veremos como pasan a ser 3 cajas azules colocadas en horizontal (*una al lado de la otra*) que cubren sólo el ancho del contenido de cada una. Ahora los `<div>` de esa página son **elementos en línea** (el tipo de representación visual que tienen los ``).

Otros tipos de display

A medida que vamos cambiando el tipo de representación de estos elementos, nos damos cuenta que es insuficiente para realizar tareas y vamos necesitando más tipos de caja.

Vamos a rellenar un poco más la tabla, con las características más importantes de las opciones que puede tomar la propiedad CSS **display**:

Tipo de caja	Características
block	Se apila en vertical. Ocupa todo el ancho disponible de su etiqueta contenedora.
inline	Se coloca en horizontal. Se adapta al ancho de su contenido. Ignora width o height .
inline-block	Combinación de los dos anteriores. Se comporta como inline pero no ignora width o height .
flex	Utiliza el modelo de cajas flexibles Flexbox . Muy útil para diseños adaptables.
inline-flex	La versión en línea (ocupa sólo su contenido) del modelo de cajas flexibles flexbox.
grid	Utiliza cuadrículas o rejillas con el modelo de cajas Grid CSS .
inline-grid	La versión en línea (ocupa sólo su contenido) del modelo de cajas grid css.
list-item	Actúa como un ítem de una lista. Es el comportamiento de etiquetas como .
table	Actúa como una tabla. Es el comportamiento de etiquetas como <table> .
table-cell	Actúa como la celda de una tabla. Es el comportamiento de etiquetas como <th> o <td> .
table-row	Actúa como la fila de una tabla. Es el comportamiento de etiquetas como <tr> .

Ocultar elementos




En la lista anterior, falta un valor de la propiedad **display**. Mediante la mencionada propiedad, es posible aplicar un valor **none** y ocultar completamente elementos que no queramos que se muestren, los cuales desaparecen por completo. Es muy útil para hacer desaparecer información cuando el usuario realiza alguna acción, por ejemplo.

Tipo de caja	Características
none	Hace desaparecer visualmente el elemento, como si no existiera.

No obstante, también existe una propiedad CSS llamada **visibility** que realiza la misma acción, con la ligera diferencia de que no sólo oculta el elemento, sino que además mantiene un vacío con el mismo tamaño de lo que antes estaba ahí. Dicha propiedad **visibility** tiene los siguientes valores posibles:

Valor	Significado
visible	El elemento es visible. Valor por defecto.
hidden	El elemento no es visible pero sigue ocupando su espacio y posición.
collapse	Sólo para tablas. El elemento se contrae para no ocupar espacio.



Utilizar **visibility:hidden** es muy interesante si queremos que un elemento y su contenido se vuelva invisible, pero siga ocupando su espacio y así evitar que los elementos adyacentes se desplacen, lo que suele ser un comportamiento no deseado en algunas ocasiones cuando se aplica **display: none**.

Otra opción interesante es utilizar la propiedad **opacity** junto a transiciones o animaciones, desplazarse desde el valor **0** al **1** o viceversa. De esta forma conseguimos una animación de aparición o desvanecimiento.

Fuente: <https://lenguajecss.com/css/maquetacion-y-colocacion/tipos-de-elementos/>



Media Queries

Las reglas **media queries** (también denominadas **MQ**) son un tipo de reglas de CSS3 que permiten crear un bloque de código que **sólo se procesará** en los dispositivos que **cumplan** los criterios especificados como **condición**.

Con **media queries** puede definir estilos completamente diferentes para diferentes tamaños de navegador.

```
@media screen and (*condición*) {  
    /* reglas CSS */  
    /* reglas CSS */  
}  
  
@media screen and not (*condición*) {  
    /* reglas CSS */  
    /* reglas CSS */  
}
```

CSS

Para ampliar:

https://www.w3schools.com/css/css_rwd_mediaqueries.asp

https://www.w3schools.com/cssref/css3_pr_mediaquery.asp

<https://lenguajecss.com/css/responsive-web-design/media-queries/>

Media Queries | Tipos de medios

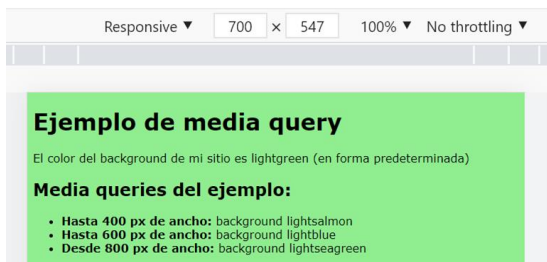
Tipo de medio	Significado
screen	Monitores o pantallas de ordenador. Es el más común.
print	Documentos de medios impresos o pantallas de previsualización de impresión.
speech	Lectores de texto para invidentes (Antes aural , el cuál ya está obsoleto).
all	Todos los dispositivos o medios. El que se utiliza por defecto .

Recordemos que con el siguiente fragmento de código HTML estamos indicando que el nuevo ancho de la pantalla es el ancho del dispositivo, por lo que el aspecto del viewport se va a adaptar consecuentemente:

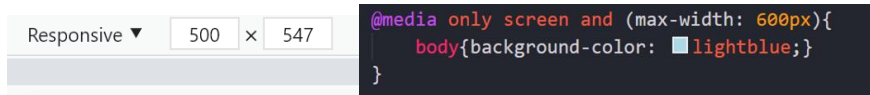
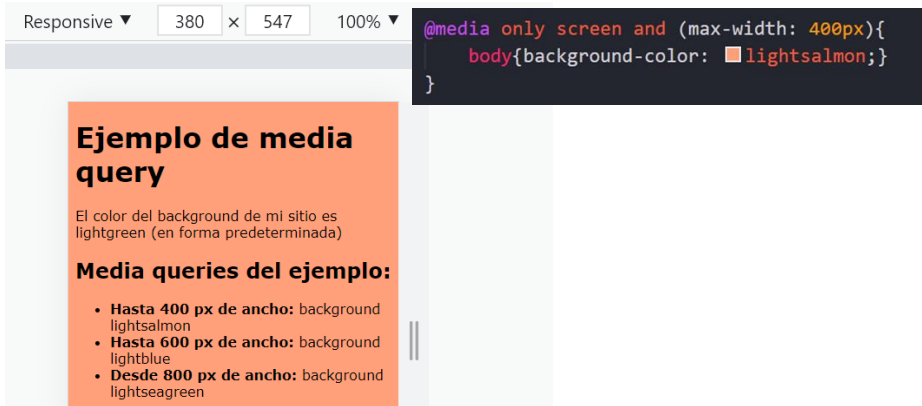
```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Media Queries | Ejemplos explicados

media-query-1.css



Background por defecto, establecido en la etiqueta **body**

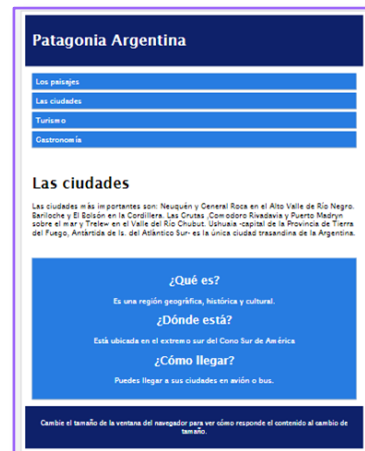


Media Queries | Ejemplos explicados

media-query-2.css



Ancho: 1200 px



Ancho: 750 px

Media Queries | Ejemplos explicados

media-query-2.css

The diagram illustrates a website layout for 'Patagonia Argentina' with various CSS classes applied to different sections. The layout is divided into a header, a main content area, and a footer.

Header: The top section is labeled 'Patagonia Argentina' and is associated with the class `<div class="header">`.

Main Content Area: This area is divided into three columns, each associated with the class `<div class="row">`.

- Left Column (Menu):** Contains a list of links: 'Los paisajes', 'Las ciudades', 'Turismo', and 'Gastronomía'. This column is associated with the class `<div class="col-3 menu">`.
- Middle Column (Cities):** Titled 'Las ciudades', it contains a paragraph about the most important cities in Patagonia. This column is associated with the class `<div class="col-6">`.
- Right Column (Info):** Titled '¿Qué es?', it contains a paragraph about the location of Patagonia. This column is associated with the class `<div class="col-3 right">`.

Footer: The bottom section is labeled 'Cambia el tamaño de la ventana del navegador para ver cómo responde el contenido' and is associated with the class `<div class="footer">`.