Frontend Developer

A principios de los '80s nace Internet, su nombre se compone de dos palabras "Interconnected" y "Network" haciendo alusión a una red inmensa de computadoras conectadas entre sí alrededor del mundo.

Tim Berners-Lee es el creador de la www (World Wide Web), que es la red con la que nos conectamos a internet, y fundó el consorcio **W3C** para estandarizar y supervisar el desarrollo de las tecnologías base de internet: HTTP, URL y HTML

* **HTTP** - Hyper Text Transfer Protocol (Protocolo de Transferencia de Hipertexto: Es la base que permite la comunicación de datos entre los dispositivos conectados en la red.
* **URL** - Uniform Resource Locator (Localizador de Recursos Uniformes): Es la dirección de un sitio web, o sea la manera en que le agregamos un nombre a un punto de la red.
* **HTML** - Hyper Text Markup Languaje (Lenguaje de Marcado de Hipertexto): Es un lenguaje de marcado (no es un lenguaje de programación) que sirve para escribir la estructura de las páginas web, otorgando múltiples recursos para dicha tarea.
* **CSS** (1994) - Cascade Style Sheets: Es un lenguaje de hojas de estilo (tampoco es un lenguaje de programación). Son una serie de reglas que describen la apariencia de una página web. Colores, tipografía, el margen y el tamaño de los elementos, inclusive adaptando las páginas web a pantallas grandes o pequeñas, conocido como Responsive Design.  
  El navegador se encarga de tomar esos estilos y agregarlos a cada elemento que esté dentro del HTML.

# Proceso de renderizado de la Web

## Los 3 componentes principales

* DOM: Document Object Model. Es una transformación del código HTML escrito por nosotros a objetos entendibles para el navegador.
* CSSOM: así como el DOM para el HTML, EL CSSOM es una representación de objetos de nuestros estilos en CSS.
* Render Tree: es la unión entre el DOM y el CSSOM para renderizar todo el código de nuestra página web.

## 5 pasos que sigue el navegador para construir las páginas web

1. Procesa el HTML para construir el DOM.
2. Procesa el CSS para construir el CSSOM.
3. El DOM se une con el CSSOM para crear el Render Tree.  
   **DOM + CSSOM = Render Tree**
4. Se aplican los estilos CSS en el Render Tree.
5. Se “pintan” los nodos en la pantalla para que los usuarios vean el contenido de la página web.

## Forma en que el navegador lee el código

1. **Bytes:** El navegador transforma el código en bytes.

|  |
| --- |
| 23 45 63 51 38 … |

1. **Character:** Luego, transforma ese código en bytes en caracteres, dependerá el tipo de caracteres de la codificación seleccionada en el meta charset (usualmente "UTF-8").

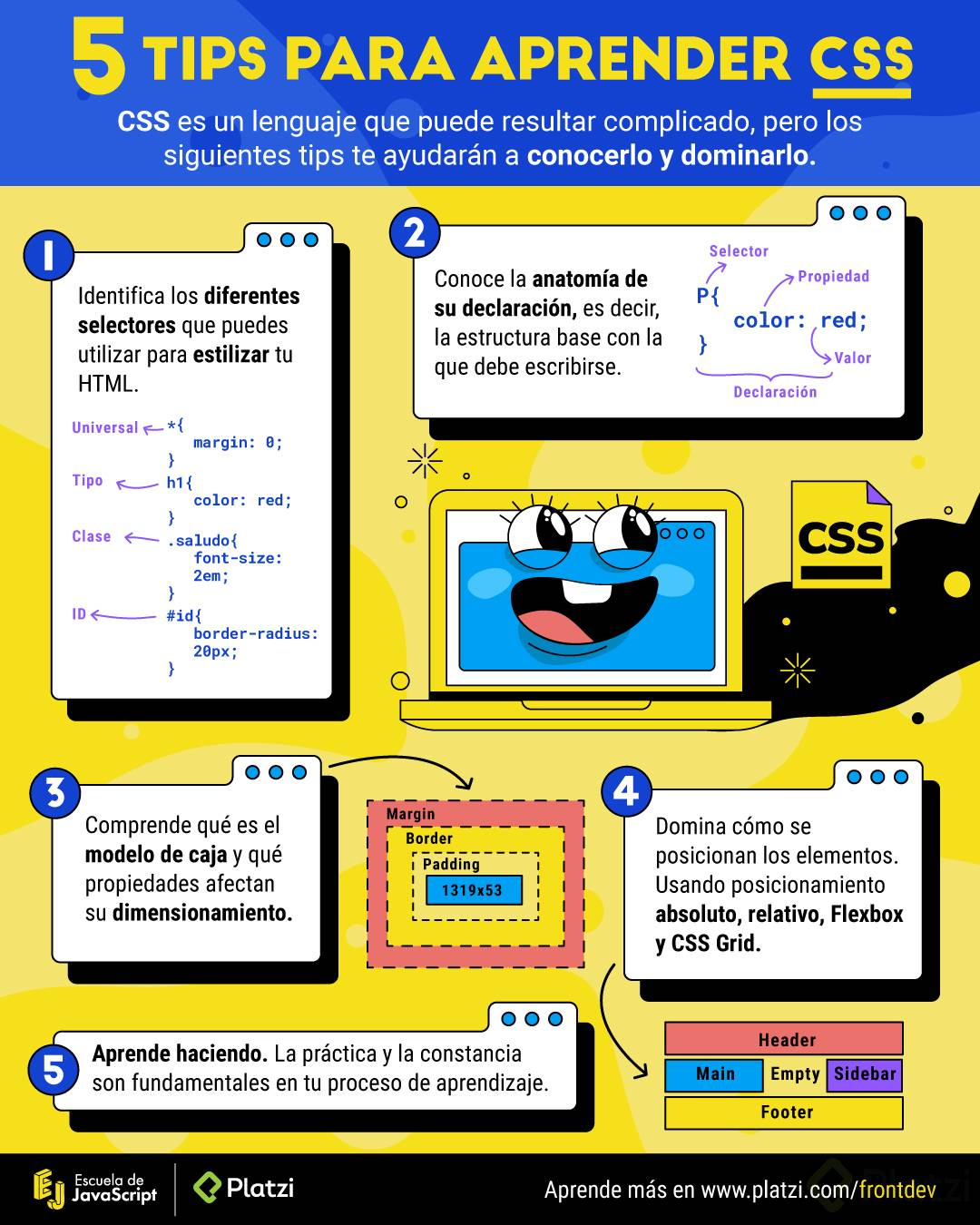
|  |
| --- |
| <html><head><link></link> … </head><body><h1>Hola</h1>… |

1. **Tokens:** Los caracteres son transformados en tokens y los ordena de acuerdo a la especificación de la W3C.

|  |
| --- |
| START TAG html  START TAG head  …  END TAG head  START TAG body  … |

1. **Nodes:** Posteriormente transforma los tokens en Nodos. Dichos nodos son objetos que puede entender, a los cuales clasifica dependiendo de si son **Tokens** o **contenido**.  
   
2. **DOM:** toma todos los elementos del paso anterior y los ubica en un árbol cuya estructura dependerá de la organización que tenga en el código.  
     
   Posteriormente el CSSOM asigna los estilos correspondientes a cada objeto del DOM.  
   

## 5 tips para aprender CSS



# Anatomía de un Elemento HTML

Los elementos HTML se componen de:

* **Etiqueta de apertura:** el nombre de la etiqueta encerrado entre símbolos de mayor o menor. Por ejemplo: <h1>.
* **Contenido**: dentro de las etiquetas podemos añadir texto u otros elementos HTML, lo que se conoce como **anidamiento**.
* **Etiqueta de cierre:** son casi iguales que las etiquetas de apertura, pero también necesitan un slash (/) antes del nombre de la etiqueta. Por ejemplo: </h1>.

Las etiquetas de apertura también pueden tener **atributos**, los cuales permiten definir características especiales para los elementos: <etiqueta atributo="valor del atributo">. Por ejemplo: <h1 class="saludo">. Los atributos siempre se ubican dentro de la etiqueta de apertura.

También existen elementos vacíos, que no tienen contenido ni etiqueta de cierre, solo etiqueta de apertura y atributos. Por ejemplo: <img src="puppy.png" alt="mi mascota">.

# Anatomía de un Documento HTML

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<style>

/\* Aquí va el código CSS \*/

</style>

<script>

// Aquí va el código JavaScript

</script>

<title>Frank</title>

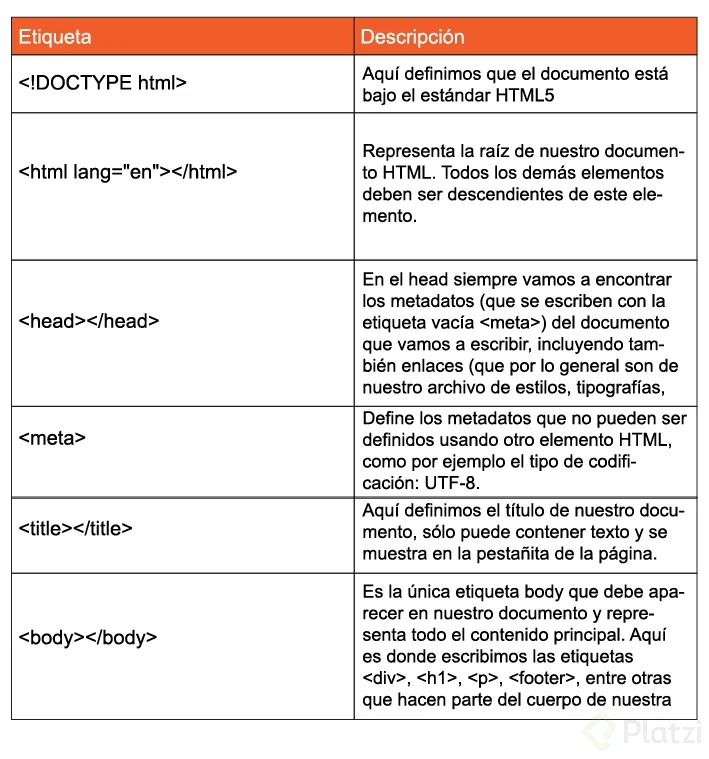
</head>

<body>

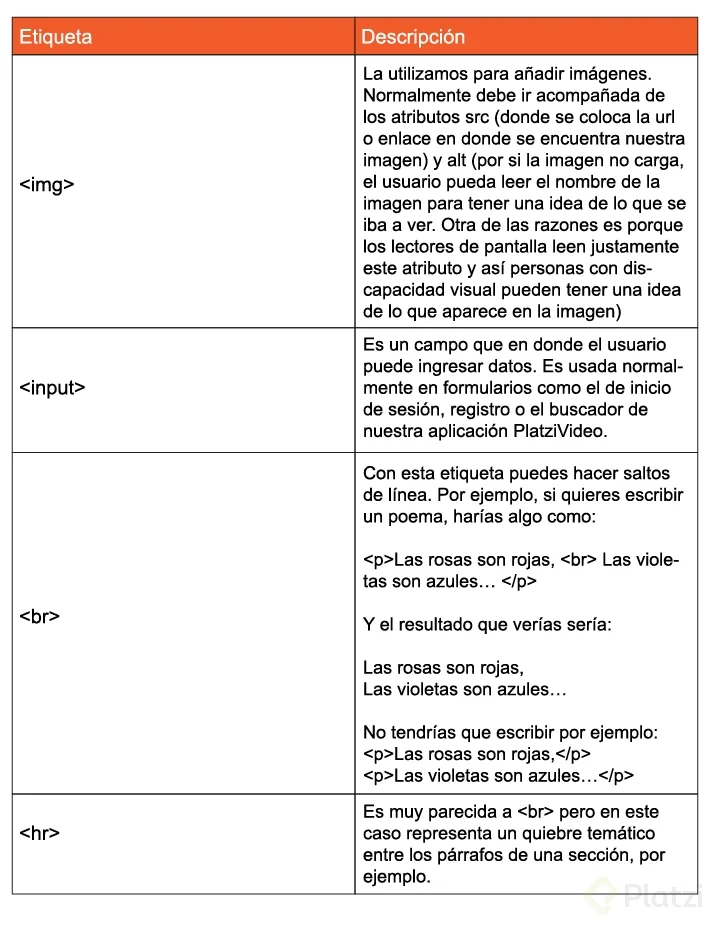
</body>

</html>

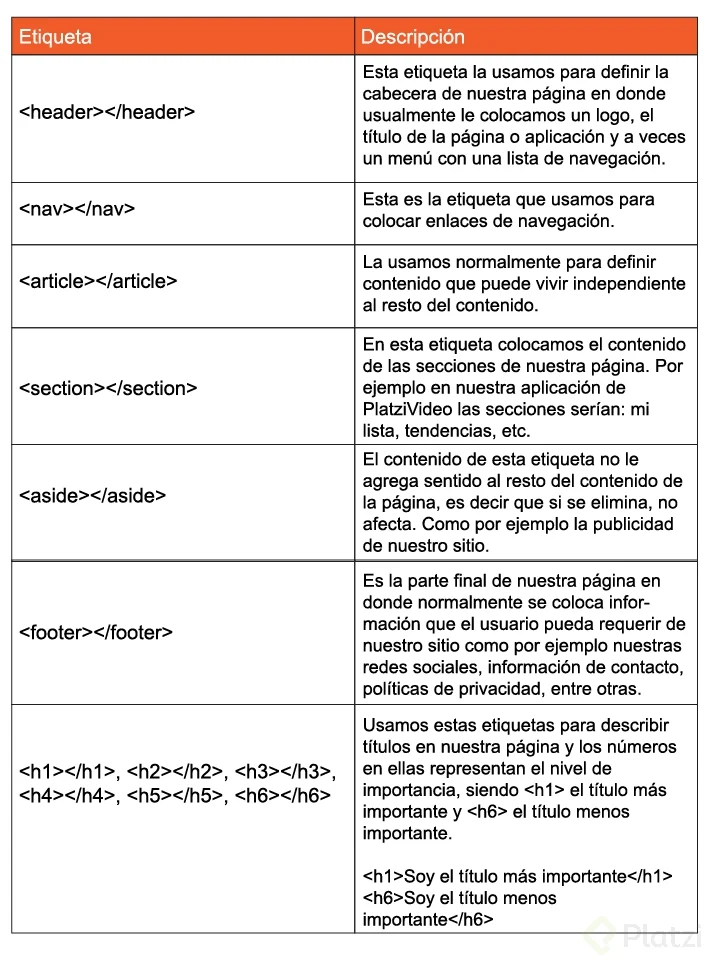
Descripción de los elementos del código:



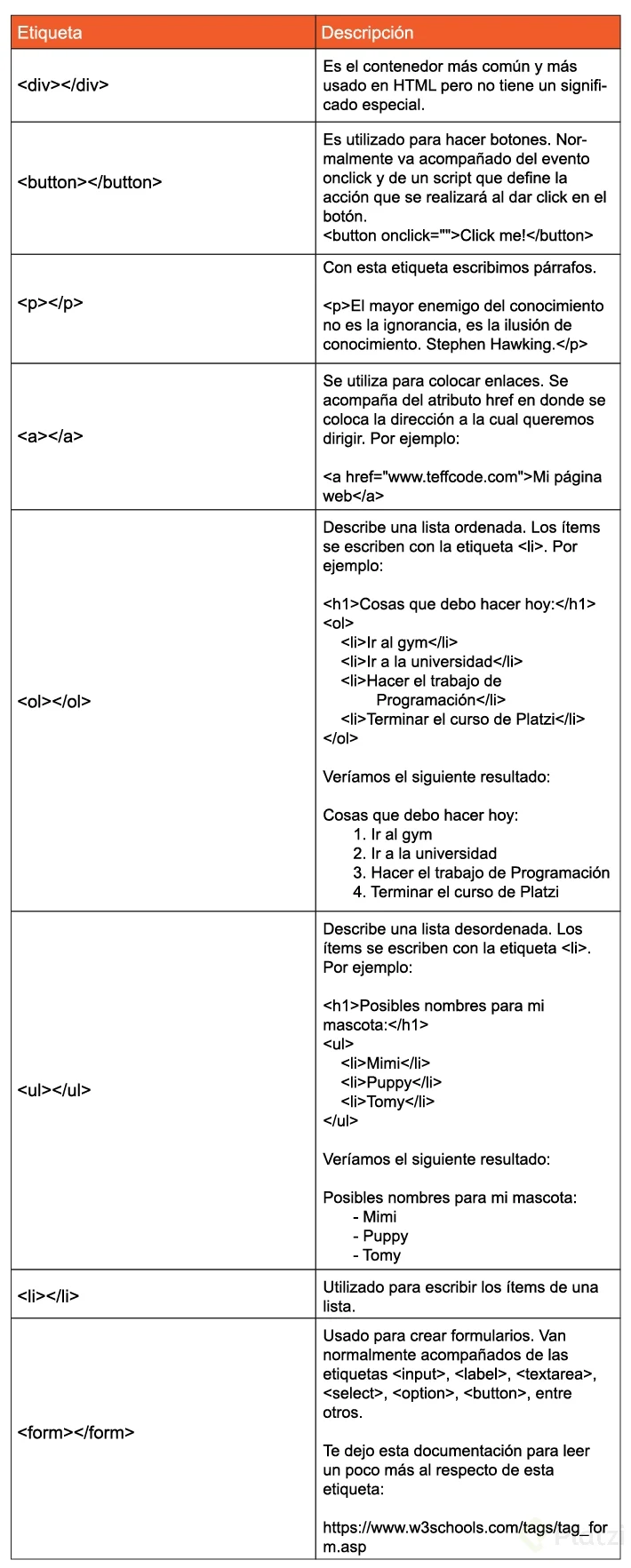
Descripción de algunos elementos vacíos:



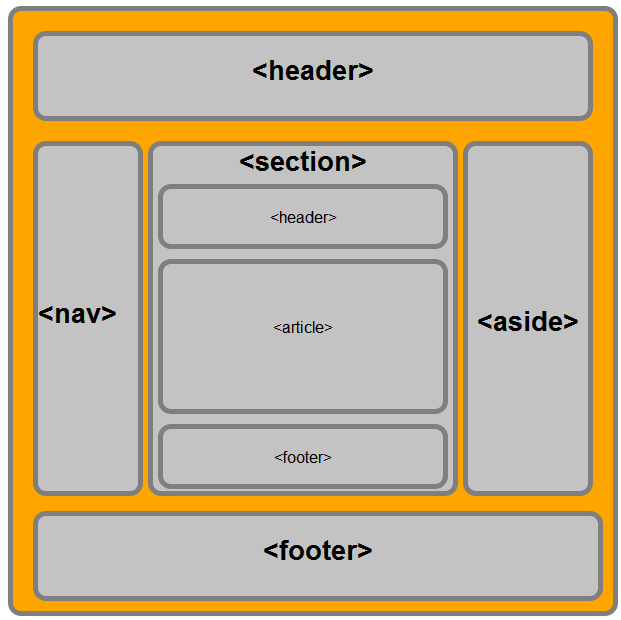
Descripción de las etiquetas semánticas para la estructura base de nuestra página:



Descripción de otras etiquetas muy usadas:



# Código Semántico



Es importante que el desarrollador tenga claro el significado de escribir código. Debe ser consciente de que la manera en la que codea tenga sentido.

La semántica HTML no es más que darle sentido y estructura a lo que se está escribiendo. Muy importante para el navegador. No todos los elementos deberían ser un div.

# Tipos de errores en HTML, debugging y servicio de validación de etiquetas

**Errores sintácticos:** Son errores de escritura en el código y evitan que el programa funcione. Pueden ser errores de tipado.

**Errores lógicos:** En estos la sintaxis es correcta, pero el código no hace lo que debería. El programa funciona, pero de forma incorrecta.

Tipos de errores y debugging en: <https://developer.mozilla.org/es/docs/Learn/HTML/Introduccion_a_HTML/Debugging_HTML>

## Servicio de validación de etiquetas - Markup Validation Service

<https://validator.w3.org/>

Esta página web toma un documento HTML como entrada, lo procesa, y genera un informe de dónde están los errores en el documento.

# Anatomía de una declaración CSS

Los estilos con CSS se componen de:

* **Selector**: son la **referencia a los elementos HTML que se quieren estilizar**. Los nombres de estas etiquetas van seguidas de una llave de apertura y otra de cierre ({}).
* **Propiedades**: son el **tipo de estilo que se quiere dar a los elementos**. Van seguidas de dos puntos (:). Las propiedades deben estar dentro de las llaves del selector que se definió anteriormente. Se puede escribir diferentes propiedades en un mismo selector.
* **Valores**: son el **estilo que se desea que tomen los elementos** HTML con respecto a una propiedad. Van seguidas de un punto y coma (;).

h1 {

color: red;

/\* "h1" es el '**selector**', "color" es la '**propiedad**' y "red" el '**valor**' \*/

}

# Tipos de selectores, pseudo-clases y pseudo-elementos

* **\*(asterisco)**: Es el selector universal. Las propiedades se aplicarán a todos los elementos de nuestro HTML.
* **Tipo**: Son selectores que se aplican a cierto elemento HTML en específico. Las propiedades se aplicarán a la etiqueta que queremos, por ejemplo p, body, html, div, etc.
* **Clase**: Si las etiquetas de HTML tienen un atributo de class se puede usar ese valor o identificador para que los cambios en el CSS afecten únicamente a ese elemento.
* **ID**: Es similar al anterior, si la etiqueta HTML tiene un ID se puede afectar solo ese elemento. No es tan recomendable trabajar con IDs
* Las **Pseudo-clases** y **Pseudo-elementos** permiten ser aún más específicos sobre qué elemento o partes de los elementos deben aplicarse los estilos.  
  Para usarlas se debe definir el selector base (por ejemplo, p) seguido de dos puntos y la pseudo-clase que se quiere estilizar (por ejemplo: p:first-child). En el caso de los pseudo-elementos se debe usar el dos puntos 2 veces (p::first-letter).

/\* Asterisco (universal) \*/

\* {

margin: 0;

}

/\* Tipo \*/

h1 {

color: red;

}

/\* Clase \*/

.saludo {

font-size: 2em;

}

/\* ID \*/

#id {

border-radius: 20px;

}

/\* Pseudo-clases \*/

p:first-child {

color: white;

}

p:last-child {

color: purple;

}

p:nth-child(2n) {

color: red;

}/\* Pseudo-elementos \*/

p::first-letter {

color: white;

}

|  |
| --- |
| No es recomendable trabajar con selectores de 'tipo' ni de 'ID' dado el exceso de especificidad.  Siempre es recomendable utilizar los selectores de 'clase'. |

# Modelo de caja

Todos los elementos HTML tienen un modelo de caja. Éste se compone, desde dentro hacia fuera: **contenido, relleno (padding), borde (border) y margen (margin)**.



Por defecto **padding y border modifican el tamaño del elemento**, si no se desea este comportamiento se utiliza la propiedad {box-sizing: border-box;}.

# Valores relativos y absolutos

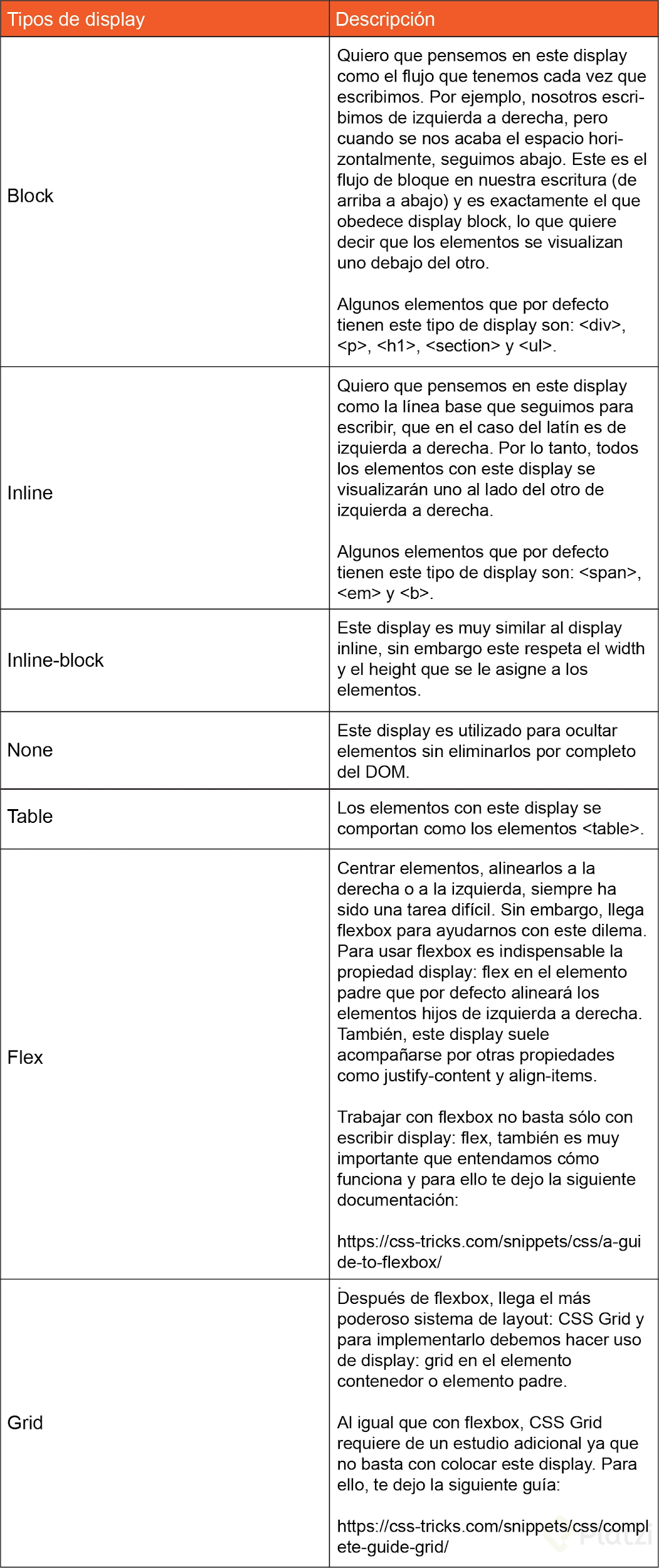
Los valores **absolutos** son: centímetros, milímetros, píxeles, pulgadas, puntos y pica (cm, mm, px, in, pt y pc respectivamente). No tienen en cuenta a nadie más, no depende de la medida de otra unidad.

Los valores **relativos** dependen de otra unidad de medida o elemento. Por ejemplo, porcentajes, vmx, em, entre otros.

Se le puede dar estilos a etiquetas HTML muy específicas indicando dónde se van a encontrar. Por ejemplo: si se le quiere dar estilos únicamente a la imagen que está dentro del header, se puede usar el selector css header img { ... }.

# Displays

Todos los elementos en CSS son cuadrados o rectángulos y aparte de eso, estos elementos tienen un comportamiento que se define a través de la propiedad display. Los display más comunes y usados son: block, inline, inline-block, none, table, flex y grid.

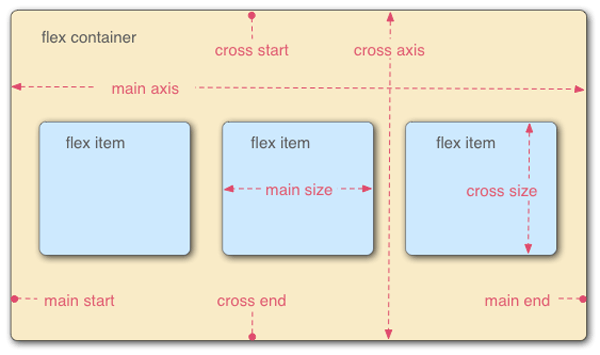


## Flexbox

Flexbox es un modelo de layout que permite organizar y ordenar los elementos en la pantalla considerando dos ejes: horizontal y vertical.

Se compone de un contenedor 'flex container' y sus hijos 'flex items'.

El container se define con la propiedad 'display: flex'

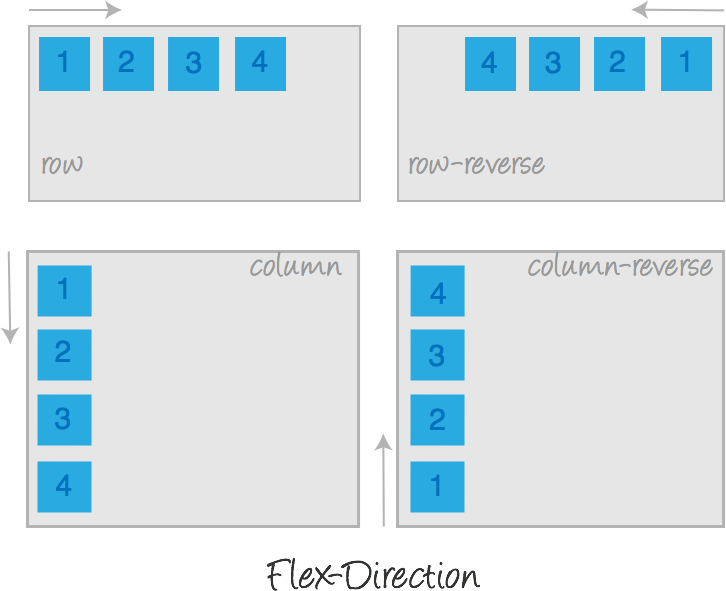


El ordenamiento se realiza entre el eje principal '**main axis**' y el eje secundario '**cross axis**'.

### Flex Direction

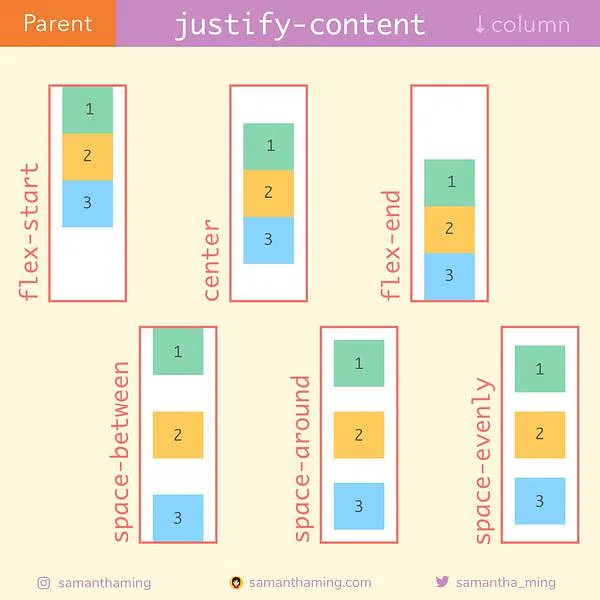
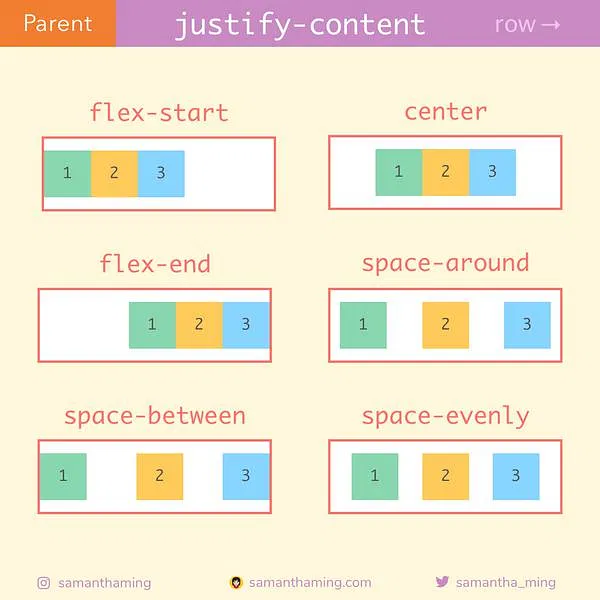
La propiedad '**flex direction**' define cuál es el main y cross axis.

Por defecto direction es row lo cual define el eje horizontal como main con un orden de izquierda a derecha y el cross axis el vertical con un orden de arriba a abajo.



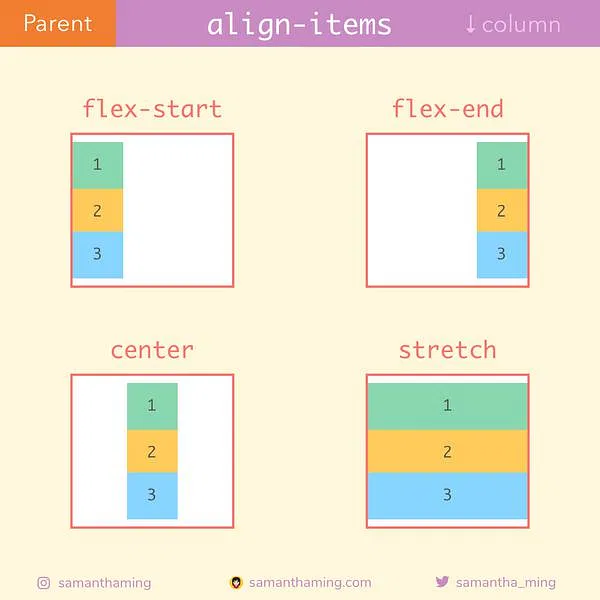
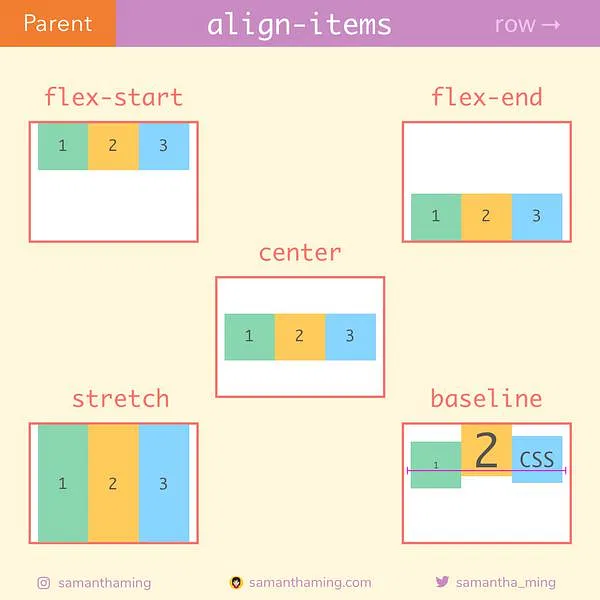
### Justify-content

Esta propiedad actúa sobre el main axis lo cual varía su comportamiento dependiendo de la propiedad flex-direction.

****

### Align items

Esta propiedad es equivalente a justify-content pero en el cross axis aunque las propiedades son algo diferentes.



# Propiedades CSS más utilizadas

**width**: Define el ancho de un elemento. Por ejemplo: width: 20px;.

**height**: Define el alto de un elemento. Por ejemplo: height: 20px;.

**background**: Puede definir el color de fondo o la url de fondo de un elemento. Por ejemplo: background: url("puppy.png");.

**background-color**: Define el color de fondo de un elemento. Por ejemplo: background-color: red;.

**color**: Define el color de nuestros textos. Estos colores los podemos escribir de 3 formas en CSS:

* Con los **nombres de los colores**. Por ejemplo: black, red, green.
* Sistema **hexadecimal**: Donde blanco se define como #FFFFFF y negro como #000000. Una página buena para sacar colores en hexadecimal es [**colorhunt.co**](https://colorhunt.co/).
* **RGB** (Red Green Blue):Cada uno de ellos es un valor de 0 a 255 que describe la intensidad de ese color. Por ejemplo, para denotar el color verde, escribimos: rgb(0, 255, 0). También a estos valores se les puede agregar una **opacidad** (transparencia) que va de 0 a 1, por ejemplo: rgba(0, 255, 0, 0.5) lo que quiere decir que el color verde lo queremos con una transparencia del 50%.

**border**: Define el tamaño, estilo y color del borde de un elemento. Por ejemplo: border: 2px solid yellow;.

**border-radius**: Define qué tan redondeado quiero mi elemento. Por ejemplo: border-radius: 20px;.

**margin**: Define el margen de un elemento. Por ejemplo: margin: 2px (lo que quiere decir que el elemento tendrá márgenes en todos sus lados de 2px).

Si solo se quiere especificar un margen se puede utilizar:

margin-right: 10px;. Y para los demás valores sería margin-top: 10px;, margin-bottom: 10px; y margin-left: 10px;.

**padding**: Define la distancia del borde de un elemento hasta su contenido. Por ejemplo: padding: 2px; (lo que quiere decir que el elemento tendrá un “margen interno” en todos sus lados de 2px).

En el siguiente caso: padding: 2px 3px 4px 5px. El **primer valor es el superior y continúa en sentido de las agujas del reloj**. Con los márgenes es igual.

Padding también tiene las propiedades: padding-right, padding-top, padding-bottom y padding-left.

**font-size**: Define el tamaño de la fuente. Por ejemplo: font-size: 20px;.

**font-family**: Define la familia tipográfica de la fuente. Por ejemplo: font-family: 'Roboto', sans-serif;.

Una página de **fuentes** muy popular es [fonts.google.com](https://fonts.google.com/).

**opacity**: Determina la transparencia del elemento. Tiene **valores entre 0 y 1, que pueden verse como un porcentaje**. Por ejemplo, para dar una transparencia del 50%: opacity: 0.5;.

**outline**: Un término algo desconocido es el esquema de los elementos HTML. Un esquema es una línea (por defecto, de color azúl) que se dibuja alrededor de los elementos que hace que “se destaquen”.

Lo anterior sucede mucho con elementos como los <input>s y los <button>s. Si no se quiere ver esa línea se define outline: none;. O sino se le puede establecer determinado estilo, color, tamaño, entre otras. Por ejemplo:

outline-style: solid;

outline-color: red;

outline-width: 5px;

**box-sizing**: Cuando se trabaja con paddings, por ejemplo, el tamaño del elemento crece. Es decir, si se tiene:

div {

background: pink;

width: 20px;

height: 20px;

}

Y se le agrega un padding de 20px, el div ya no tiene un ancho y un alto de 20px, sino de 40px cada uno. Esto implica que el padding hizo que creciera el elemento. **Si no se quiere que el padding afecte el tamaño original, hay que agregar la propiedad box-sizing** para que el tamaño total del elemento incluya el padding también y no se vea afectado por él.

**transition**: Las transiciones CSS permiten cambiar los valores de las propiedades sin problemas durante una duración determinada. **Una transición NO es una animación. Una transición va de un punto A a un punto B sin interrupciones o saltos en medio.**

Links con documentación sobre las propiedades y valores disponibles:

* <https://www.w3schools.com/css/css3_transitions.asp>.
* <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions>
* <https://css-tricks.com/almanac/properties/t/transition/>

**animation**: Esta propiedad permite animar elementos.

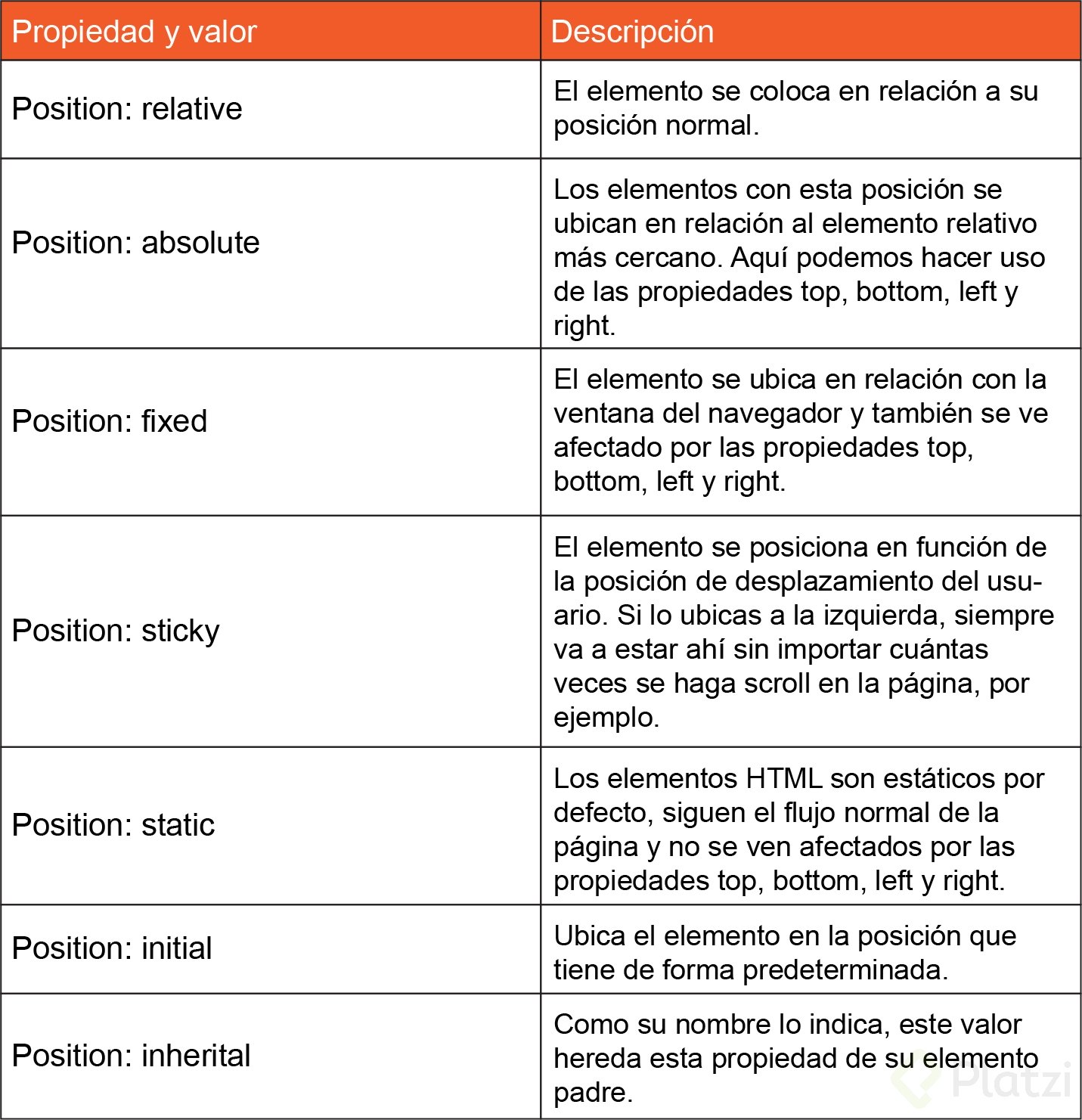
Links con documentación sobre las propiedades y valores disponibles:

* <https://www.w3schools.com/css/css3_animations.asp>
* <https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Animations/Using_CSS_animations>
* <https://css-tricks.com/almanac/properties/a/animation/>

# Posicionamiento en CSS

El posicionamiento en CSS es una de las cosas más importantes, pues establece cómo van a estar ubicados los elementos en la pantalla.

Las propiedades a utilizar son: top, bottom, left y right. **Solo funciona si la propiedad position está establecida**. Por ejemplo, para ubicar un elemento div completamente a la derecha: div { position: absolute; right: 0px; }.

La propiedad position tiene 7 valores diferentes: relative, absolute, fixed, sticky, static, initial e inherit.

# Arquitecturas CSS

Objetivos:

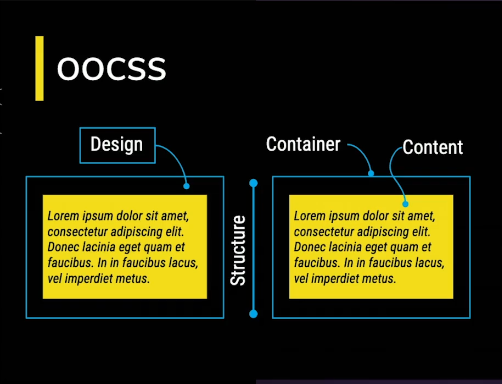
* **Predecible**: Escribir reglas claras y que se cumplan tal cual lo establecido.
* **Reutilizable**: No escribir código redundante.
* **Mantenible**: Que sea fácil de leer y adaptarnos a los estándares.
* **Escalable**: Que pueda crecer fácilmente pero sin afectar el rendimiento.

Buenas practicas:

* **Establecer reglas de arquitectura** y comunicarlas a todo el equipo.
* **Explicar la estructura base** o dar los fundamentos del proyecto a un nuevo integrante.
* **Evitar hojas de estilo muy extensas**
* Tener una buena **documentación** explicando ciertos aspectos del código.

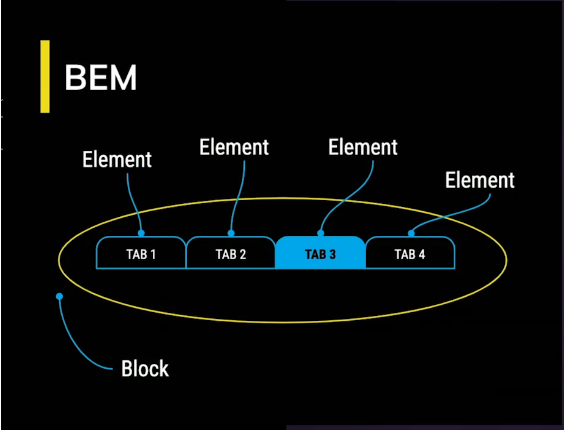
## OOCSS, BEM, SMACSS, ITCSS y Atomic Design

**OOCSS**: CSS orientado a objetos, separa el diseño del contenido



<https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/>

**BEM**: Block Element Modifier. Separa los bloques, los elementos y los modificadores.



<!-- Bloque: header - Elemento: button - Modifier: red/yellow -->

<header class="header">

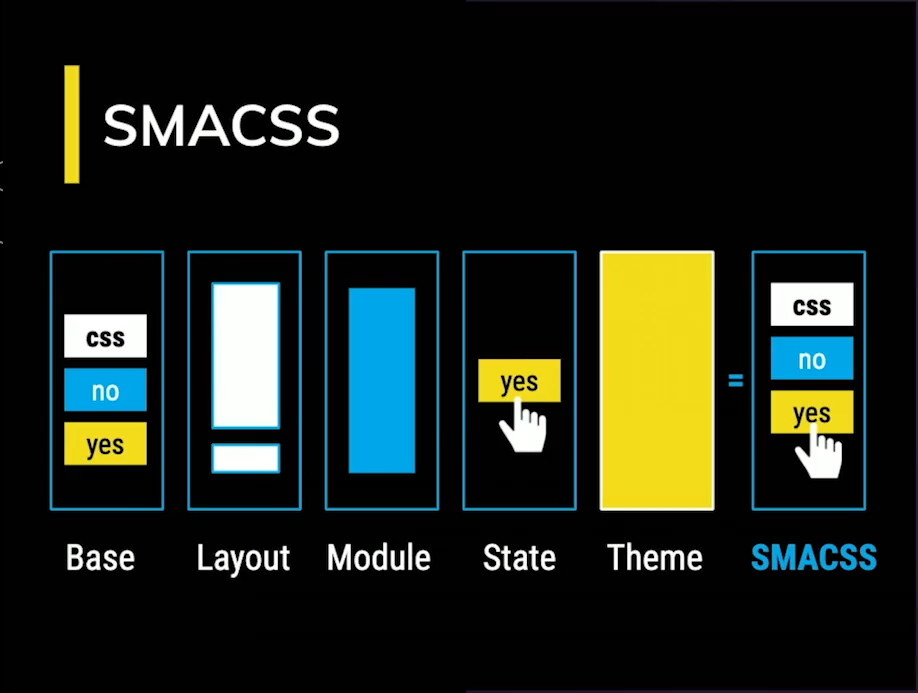
<button class="header\_\_button--red">RED</button>

<button class="header\_\_button--yellow">YELLOW</button>

</header>

<http://getbem.com/introduction/>

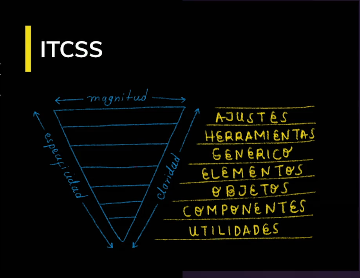
**SMACSS**: Arquitectura de CSS Escalable y Modular



1. Dividir el CSS en componentes base`que se utilizarán en toda la aplicación.
2. Layout son los elementos que se utilizan en la página por única vez. P.ej: header o footer.
3. Module son los componentes que se utilizarán más de una vez.
4. State p.ej sería el color de un botón, primero está de un color y al pulsarlo cambia a otro color.
5. Themes en caso que la aplicación tenga distintos estilos de visualización, los mismos deben poder diferenciarse estando estando separados en el código.

<http://smacss.com/>

**ITCSS**: Triángulo invertido de CSS

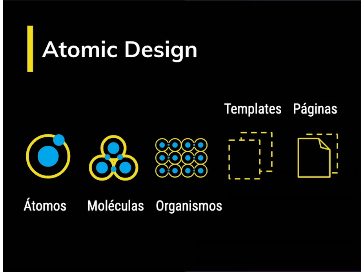


Divide el archivo de CSS en ciertas partes para que no se combinen entre sí.

**Especificidad**: Hay elementos o clases que tienen mucho más peso que otros.

<https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/>

**Atomic Design**: Al diseñar este modelo se basó en la química.



Átomos serían elementos como los botones, Moléculas podría ser un conjunto de botones y así se va creciendo en agrupaciones formando Organismos, Templates y Páginas.

<https://bradfrost.com/blog/post/atomic-web-design/>

# Componentes

Un componente, tanto en diseño como desarrollo web, es un elemento muy pequeño que tiene la capacidad de ser reutilizado en diferentes partes de una aplicación. Por ejemplo: botones, iconos, cards, entre otras.

# Mediaqueries vs Mixin

Un Mixin se usa típicamente con herencia múltiple. Así que, en ese sentido, "no hay diferencia".

El detalle es que un Mixin rara vez es útil como un objeto independiente.

Por ejemplo, si se tiene un Mixin llamado "ColorAndDimension", que añade una propiedad de color, el ancho y la altura.

Ahora, se podría añadir ColorAndDimension a una "Clase de Forma", una "Clase de Sprite", una "Clase de Coche", etc. Y todos ellos tendrán la misma interfaz (get/setColor, get/setAltura/Ancho, etc.)

Así que, en el caso genérico, una mezcla en la herencia de IS. Pero se puede argumentar que es una cuestión del papel de la clase en el dominio general en cuanto a si un Mixin es una clase "primaria" o simplemente un Mixin.

Un Mixin puede ser considerado, en la jerga moderna de hoy en día, una Interfaz con una Implementación asociada. En realidad es simplemente una herencia múltiple simple, antigua y cotidiana usando una clase simple, antigua y cotidiana. Sólo resulta ser una aplicación específica de MI.

La mayoría de los lenguajes no dan un estatus especial a Mixin, es sólo una clase que fue diseñada para ser "mezclada", en lugar de ser usada de forma independiente.

Fuente: <https://stackoverflow.com/questions/860245/mixin-vs-inheritance#860312>