**¿Qué son y para qué nos sirven HTML y CSS?**

**HTML (Hyper Text Markup Language)**: Es un lenguaje de marcado usado para decirle a tu navegador cómo estructurar las páginas web que visitas. No es un lenguaje de programación.

**CSS (Cascade Style sheets)**: Es un lenguaje que nos permite crear páginas web con un diseño agradable para los usuarios. Tampoco es un lenguaje de programación.

**DOM, CSSOM, Render Tree y el proceso de renderizado de la Web**

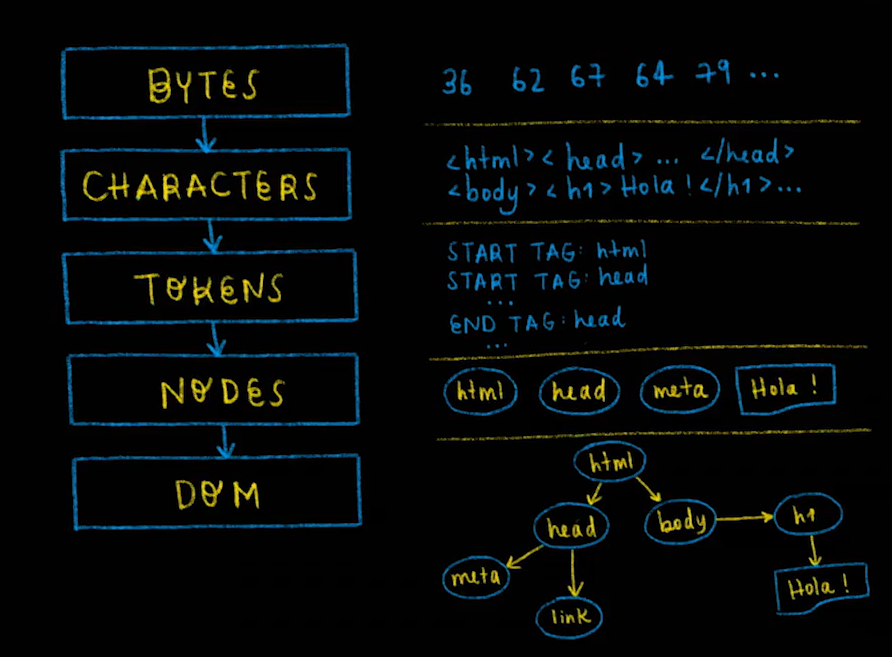
**DOM**: Document Object Model. Es una transformación del código HTML escrito por nosotros a objetos entendibles para el navegador.

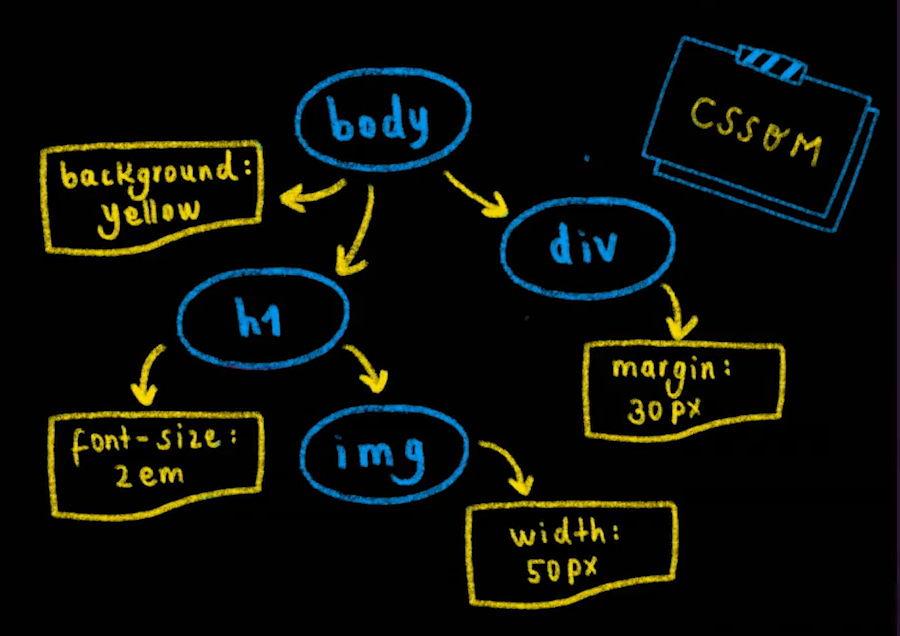
**CSSOM**: así como el DOM para el HTML, EL CSSOM es una representación de objetos de nuestros estilos en CSS.

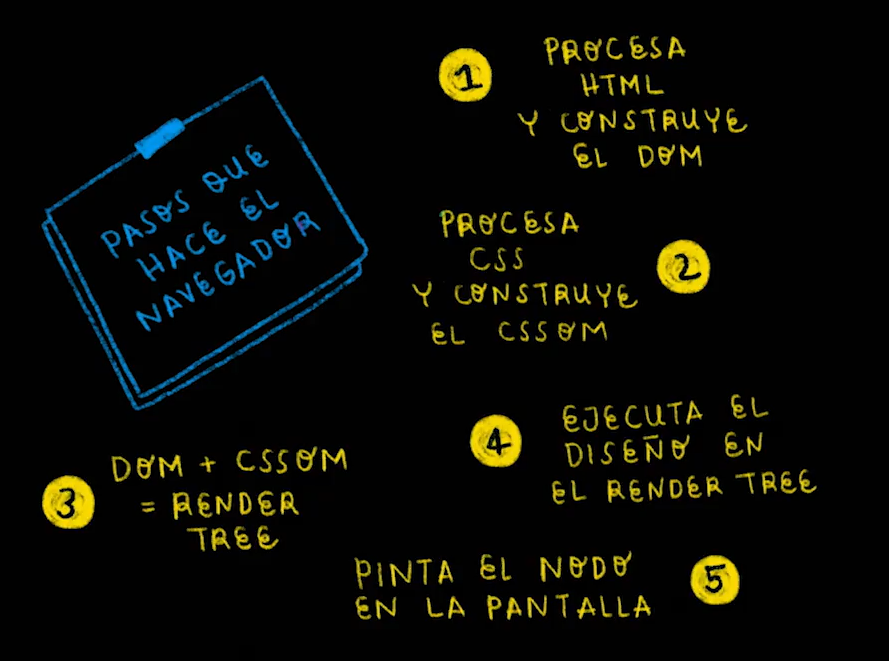
**Render Tree**: es la unión entre el DOM y el CSSOM para renderizar todo el código de nuestra página web.

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.
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.









**Anatomía de un Elemento HTML: Atributos, Anidamiento y Elementos vacíos**

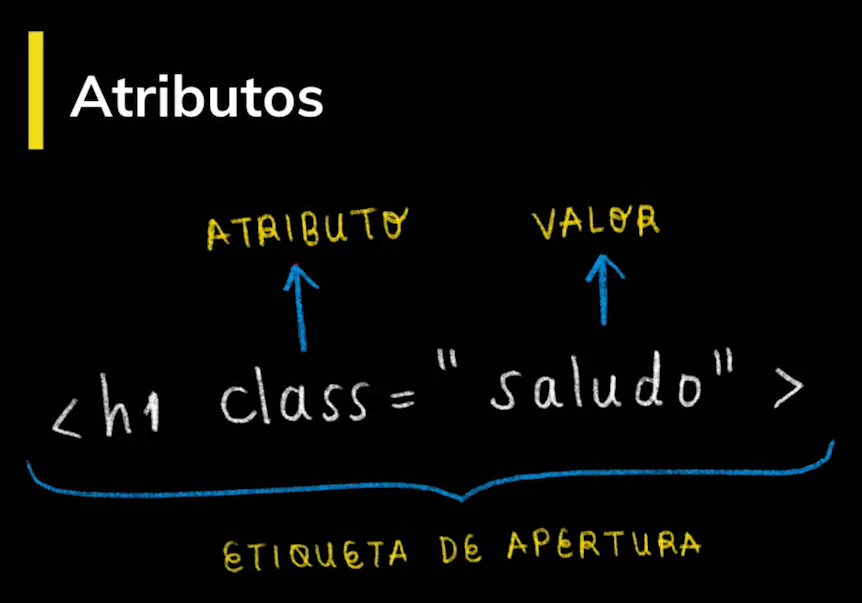
Nuestros elementos HTML se componen de:

* **Etiqueta de apertura**: el nombre de nuestra etiqueta encerrado entre símbolos de mayor o menor. Por ejemplo: **<h1>**.
* **Contenido**: dentro de nuestras etiquetas podemos añadir texto u otros elementos HTML, lo que conocemos 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>**.

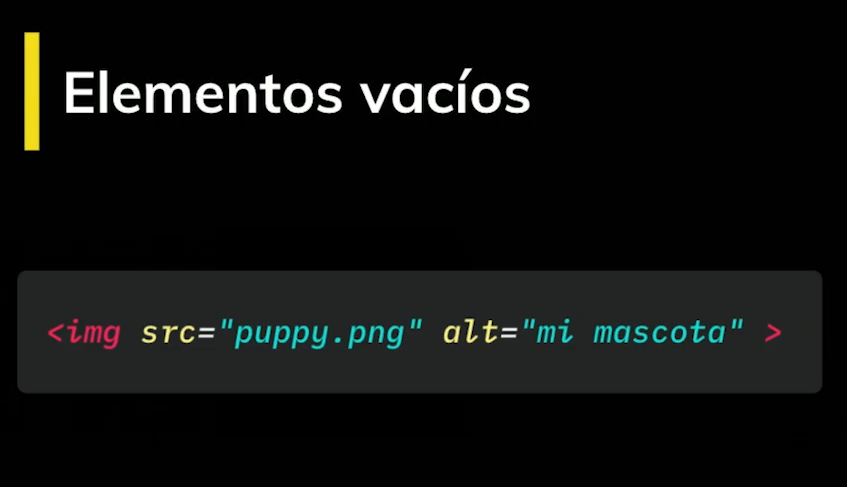
Las etiquetas de apertura también pueden tener atributos. Los atributos nos permiten definir características especiales para nuestros elementos: **<etiqueta atributo=""valor del atributo"">**. Por ejemplo: **<h1 class=""saludo"">**.

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

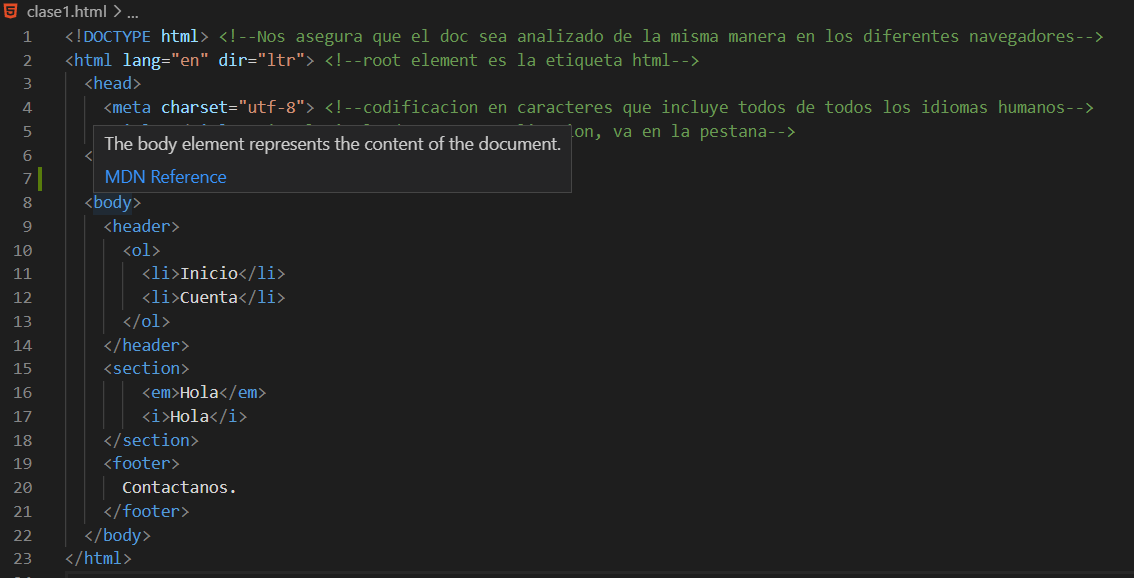








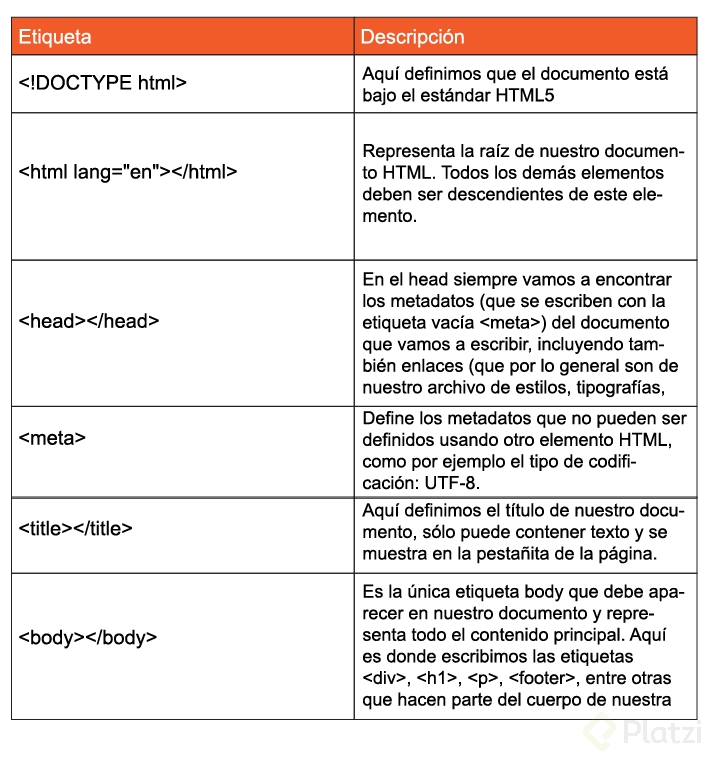
**Anatomía de un Elemento HTML: DOCTYPE, html, head y body**



**Funciones de las etiquetas HTML más importantes**



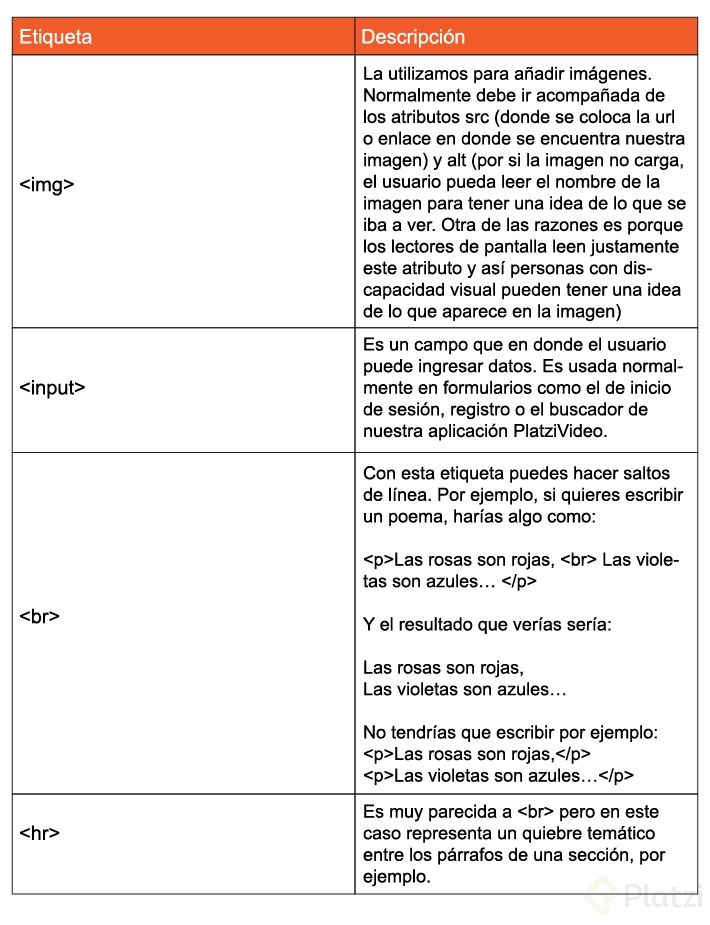
Veamos la descripción de estos elementos:



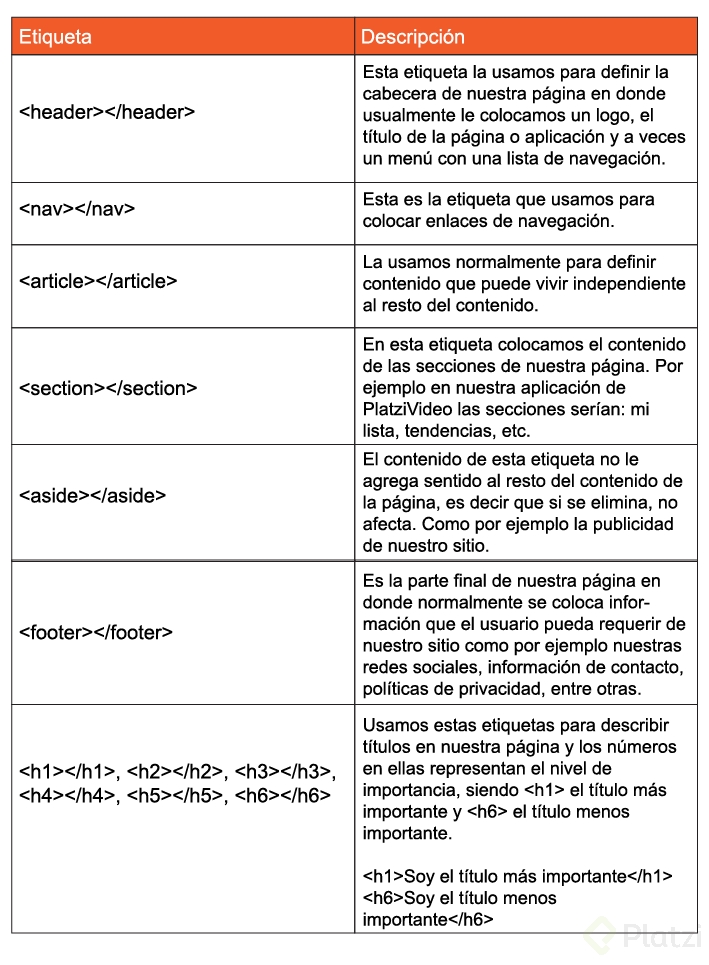
Si por ejemplo queremos añadir en nuestro documento estilos o JavaScript, lo hacemos con las etiquetas <style> y <script> respectivamente, que a su vez deben ir dentro de la etiqueta raíz <html>, algo así:



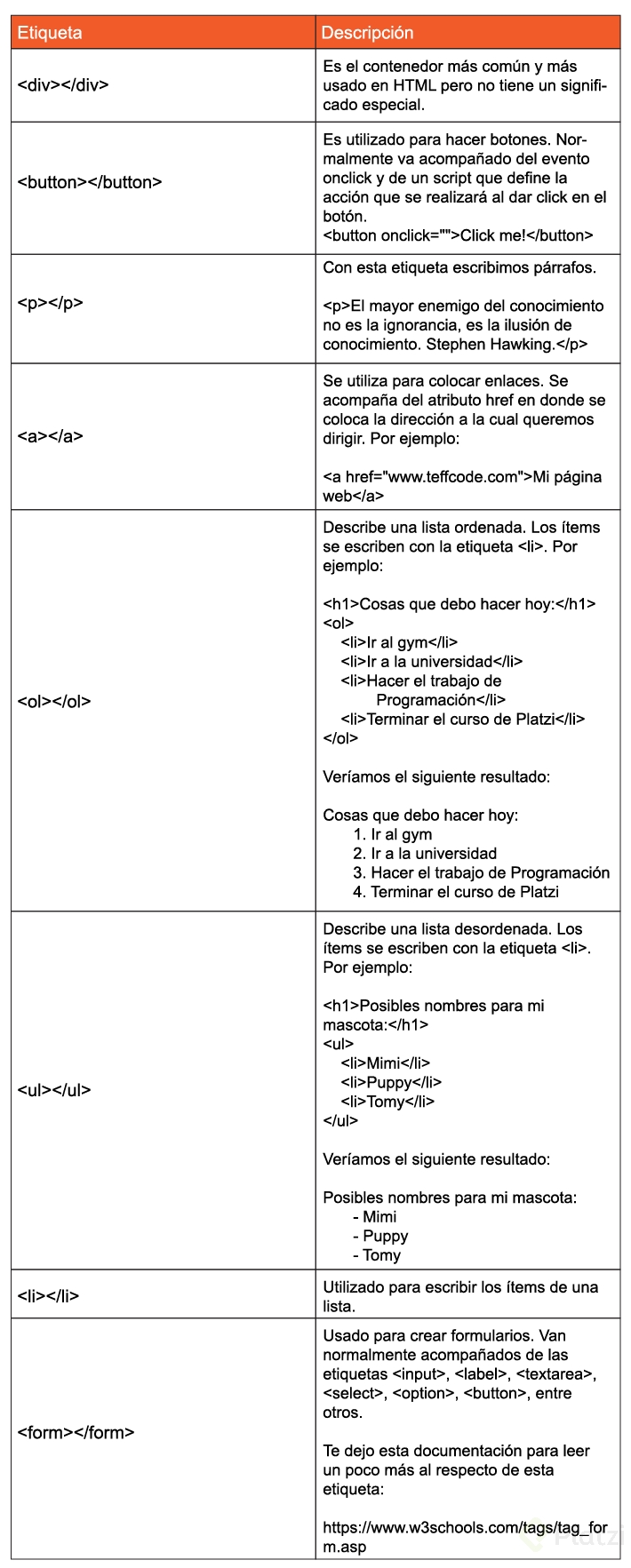
Descripción de algunos elementos vacíos:



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



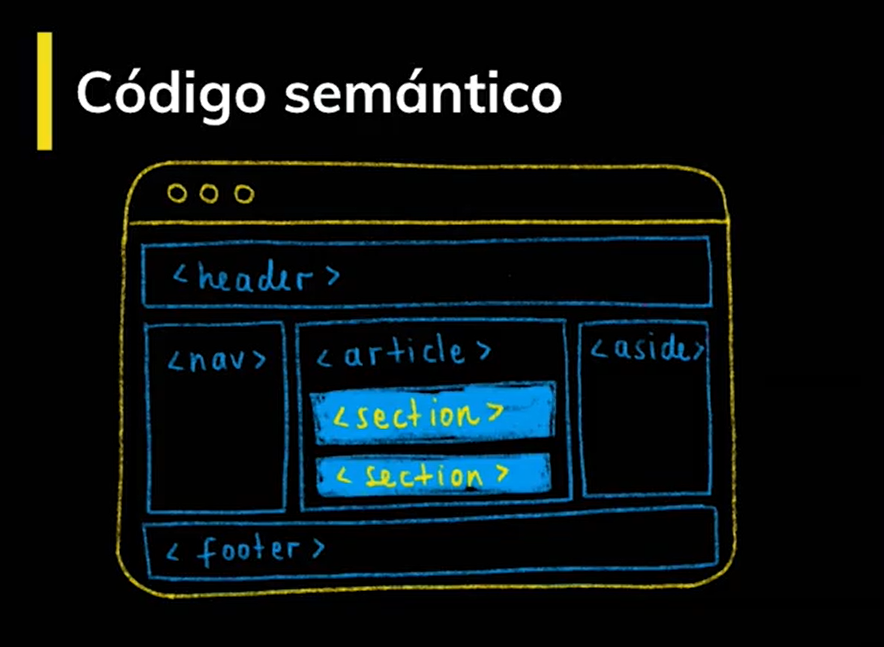
Descripción de otras etiquetas muy usadas:



**La importancia del código semántico**

Es importante que como desarrolladores tengamos claro el significado de escribir código. Debes ser consciente de que la manera en la que codeas tenga sentido.

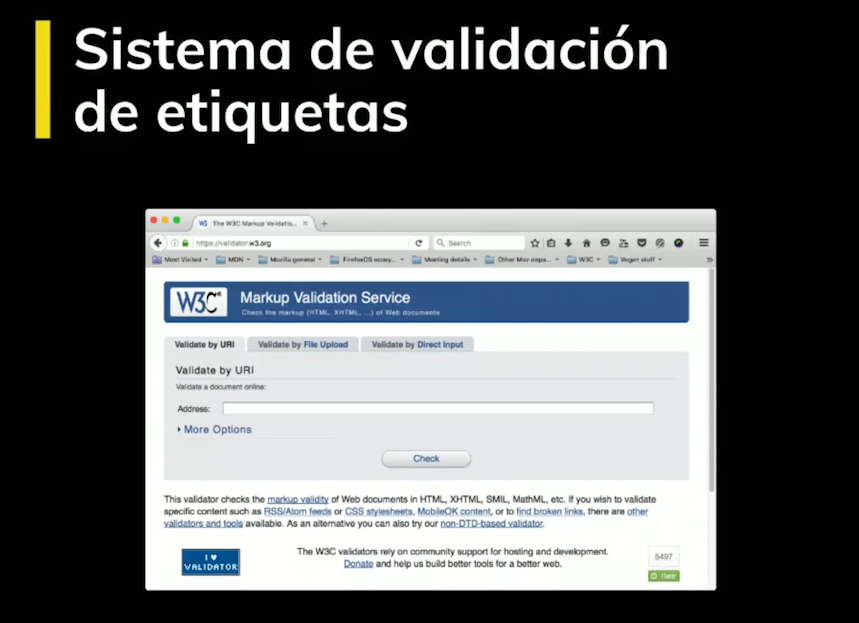
La semántica HTML no es más que darle sentido y estructura a lo que estas escribiendo. Muy importante para el navegador. No todos los elementos deberían ser un div.



**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.

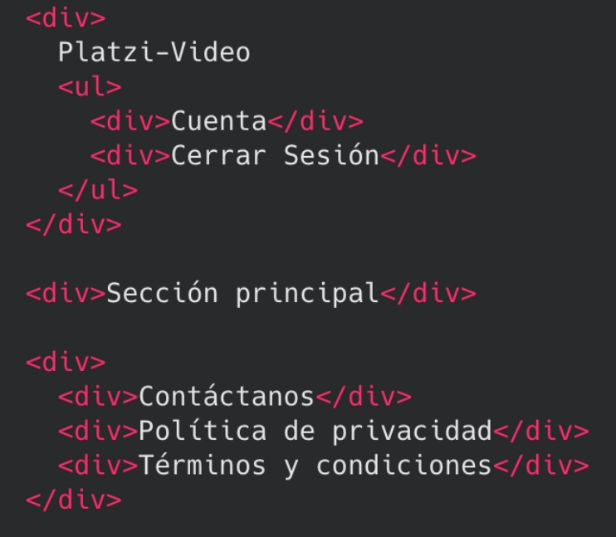


<https://validator.w3.org/>

# Reto 1: Organiza el siguiente bloque de código de forma semántica.

Ya conocimos la importancia de hacer nuestro HTML de forma semántica. Así que, aquí tengo un reto para ti.

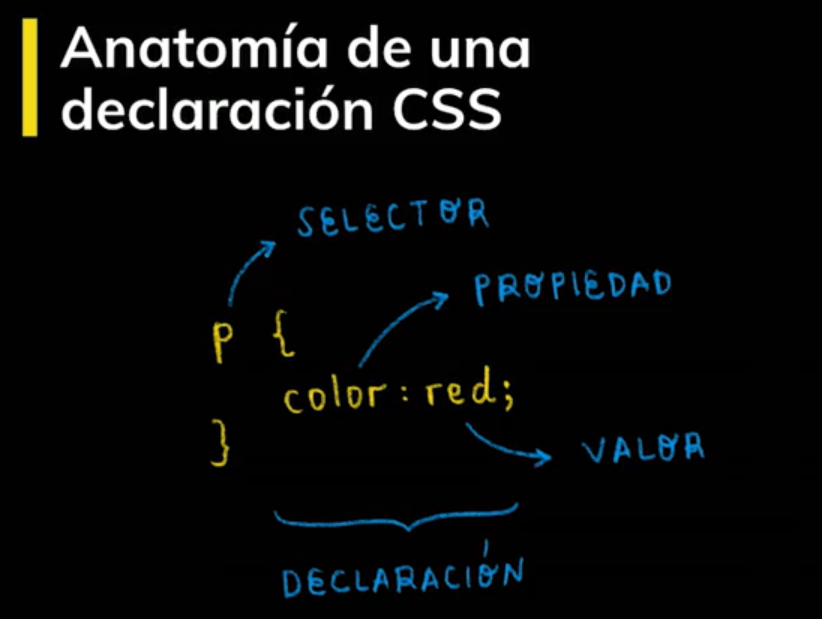
A continuación te mostraré un código HTML y la idea es que tu lo organices de forma semántica, es decir, que uses las etiquetas más adecuadas.



**Anatomía de una declaración CSS: Selectores, Propiedades y Valores**

Nuestros estilos con CSS se componen de:

* **Selector**: son la referencia a los elementos HTML que queremos estilizar. Los nombres de estas etiquetas van seguidas de una llave de apertura y otra de cierre (**{}**). Por ejemplo: **h1 {}**.
* **Propiedades**: son el tipo de estilo que queremos darle a nuestros elementos. Van seguidas de dos puntos (**:**). Las propiedades deben estar dentro de las llaves del selector que definimos anteriormente. Podemos escribir diferentes propiedades en un mismo selector. Por ejemplo: **h1 { color: }**.
* **Valores**: son el estilo que queremos que tomen nuestros elementos HTML con respecto a una propiedad. Van seguidas de un punto y coma (**;**). Por ejemplo: **h1 { color: red; }**.



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

\***(asterisco)**: Es el selector universal. Las propiedades se aplicaran a todos los elementos de nuestro HTML.

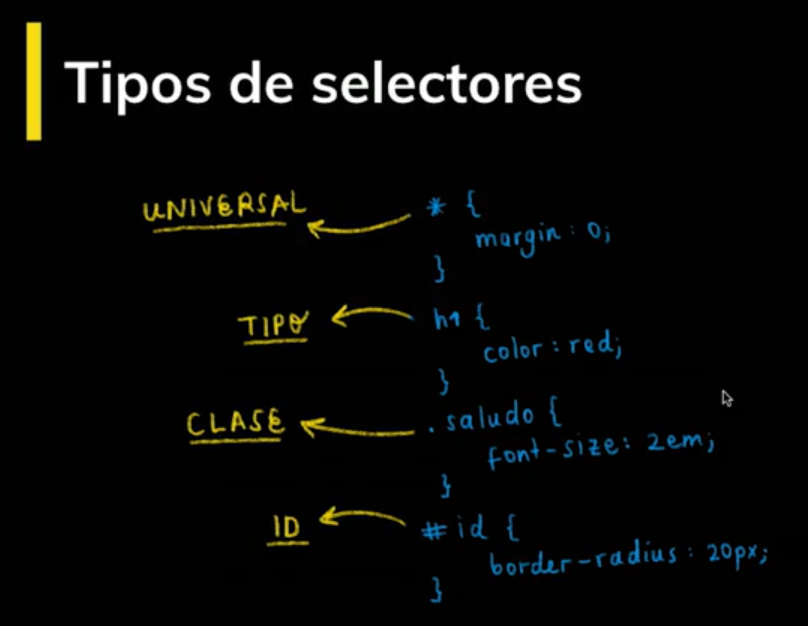
**Tipo**: Son selectores que se aplican a cierto elemento HTML en específico. Las propiedades se aplicaran a la etiqueta que queremos, por ejemplo p, body, html, div, etc.

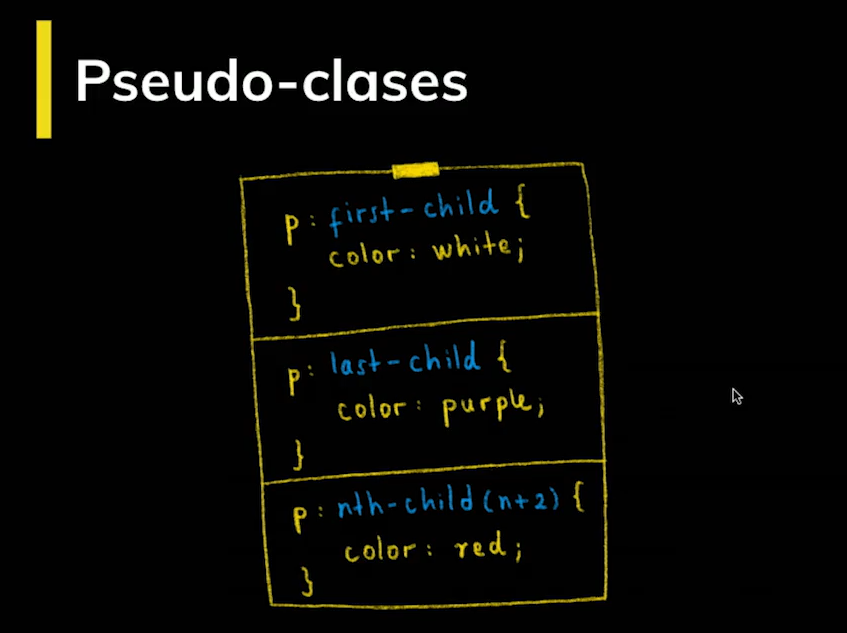
**Clase**: Si nuestras etiqueta de HTML tienen un atributo de class podemos usar ese valor o identificador para que los cambios en el CSS afecten únicamente a ese elemento.

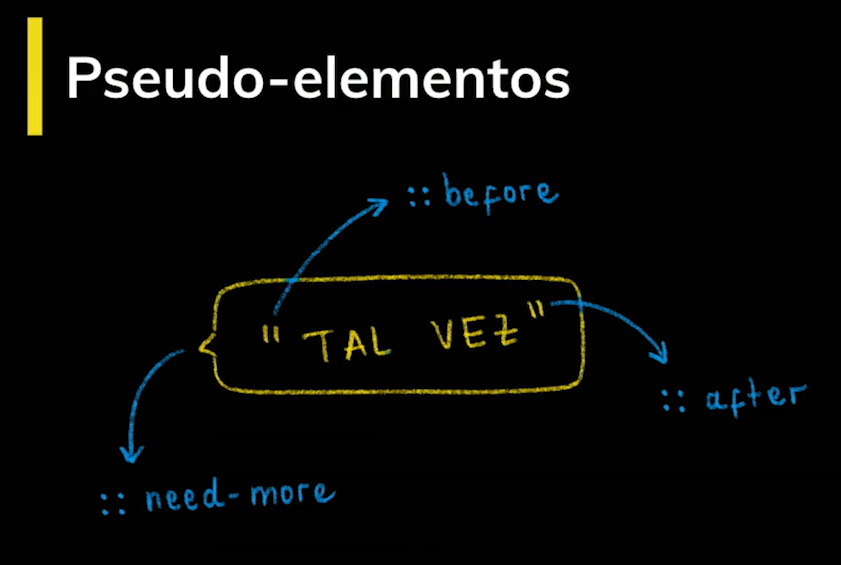
**ID**: Es similar al anterior, si la etiqueta HTML tiene un ID podemos afectar solo ese elemento.

Las **Pseudo-clases** y **Pseudo-elementos** nos permiten ser aún más específicos con qué elemento o partes de nuestros elementos deben recibir los estilos.

Para usarlas debemos definir el selector base (por ejemplo, **p**) seguido de dos puntos y la pseudo-clase que queremos estilizar (por ejemplo: **p:first-child**). En el caso de los pseudo-elementos debemos usar el dos puntos 2 veces (**p::first-letter**).





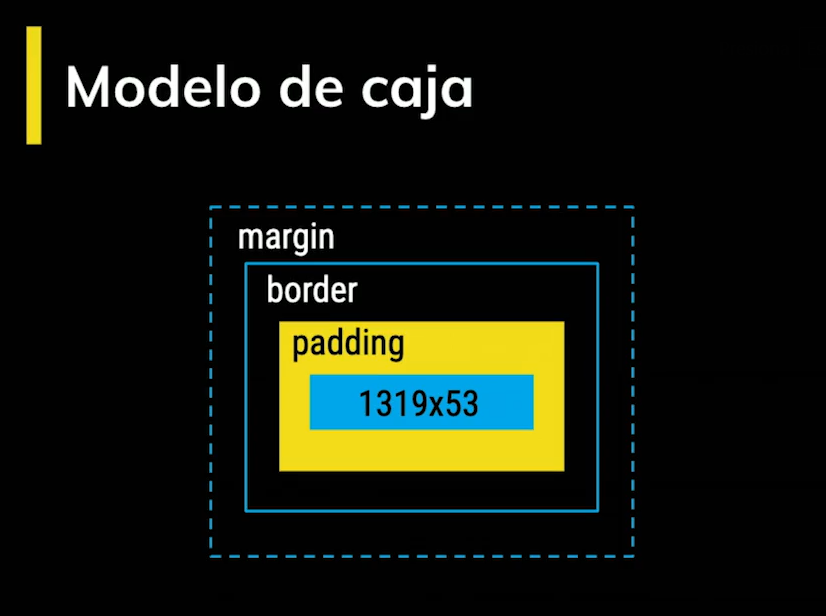


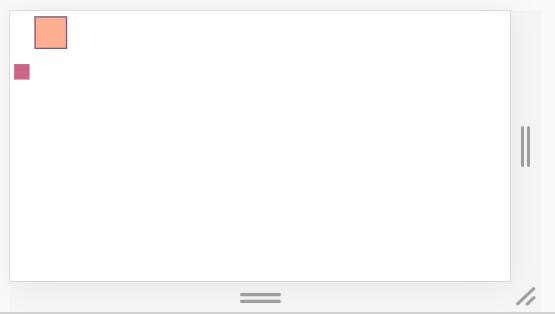
<https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes>

<https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-elements>

**Modelo de caja**

Todos los elementos de HTML tienen un modelo de caja y está compuesto por cuatro elementos: **contenido**, **padding**, **border**, **margin**.





**Valores relativos y absolutos**

Los valores **absolutos** son, por ejemplo, centímetros, milímetros, pixeles y pulgadas. Se llaman de esta forma porque no tienen en cuenta a nadie más, no depende de la medida de otra unidad.

Los valores **relativos**, llevan este nombre porque depende de otra unidad de medida o elemento. Por ejemplo, porcentajes, vmx, em, entre otros.

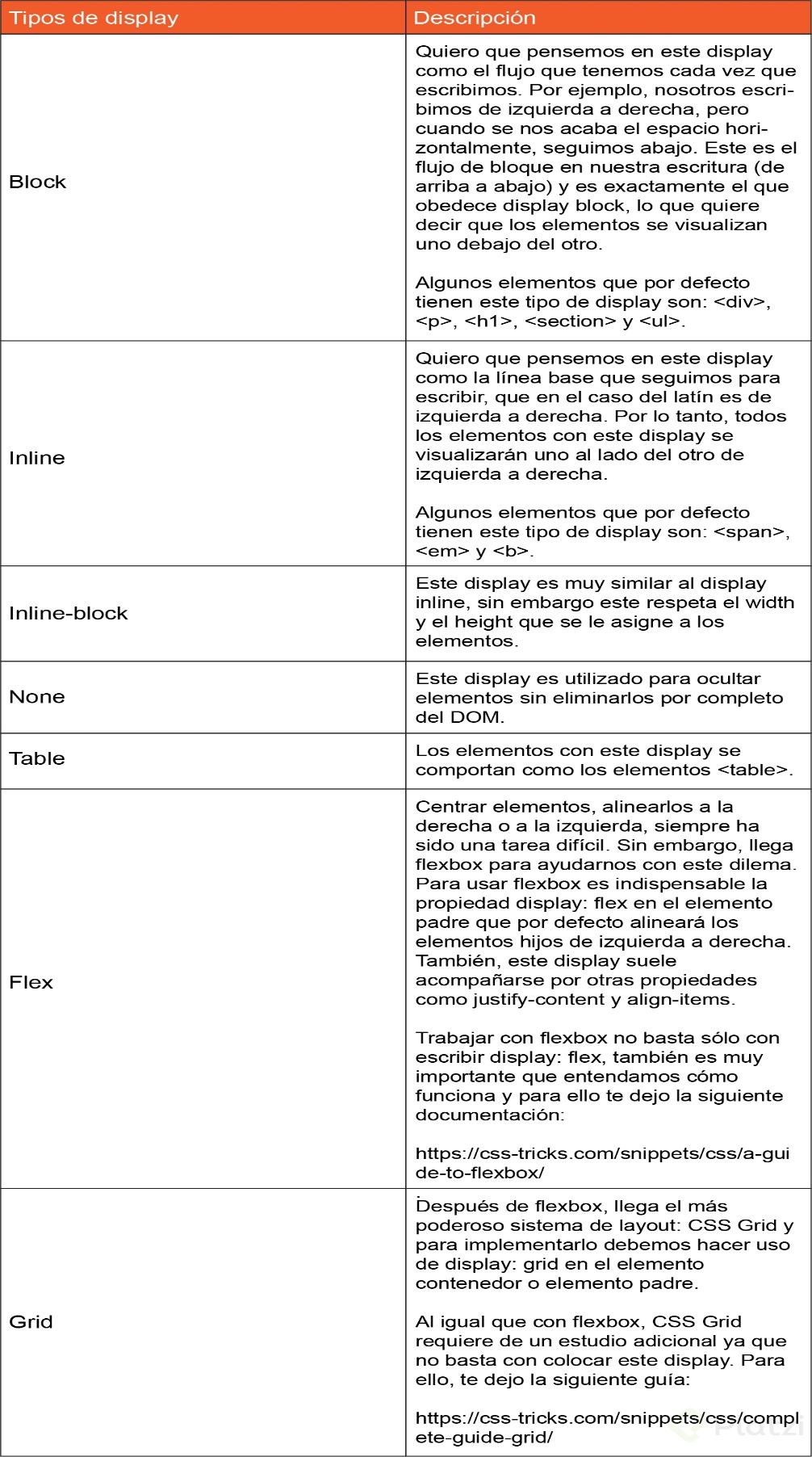
Recuerda que podemos darle estilos a etiquetas HTML muy específicas indicando dónde se van a encontrar. Por ejemplo: si queremos darle estilos únicamente a la imagen que está dentro del header, podemos usar el selector css **header img { ... }**.

No olvides resolver el desafío: crear tu propio header con las etiquetas y estilos que más te gusten para compartirlo en la sección de discusiones.

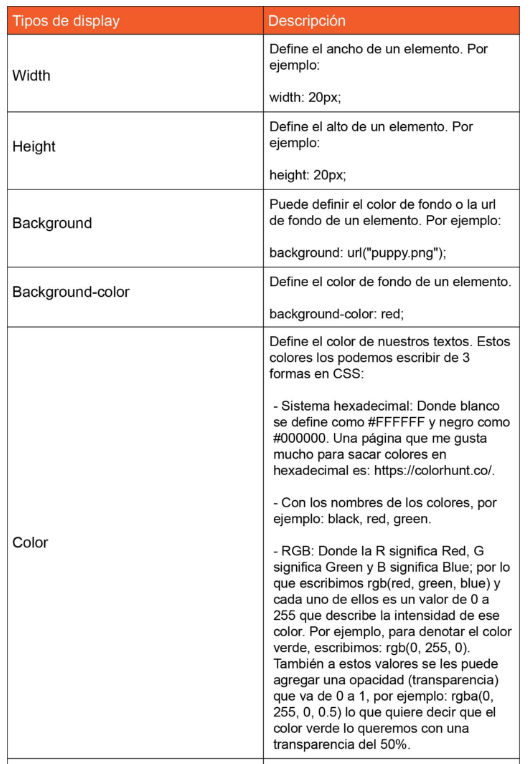


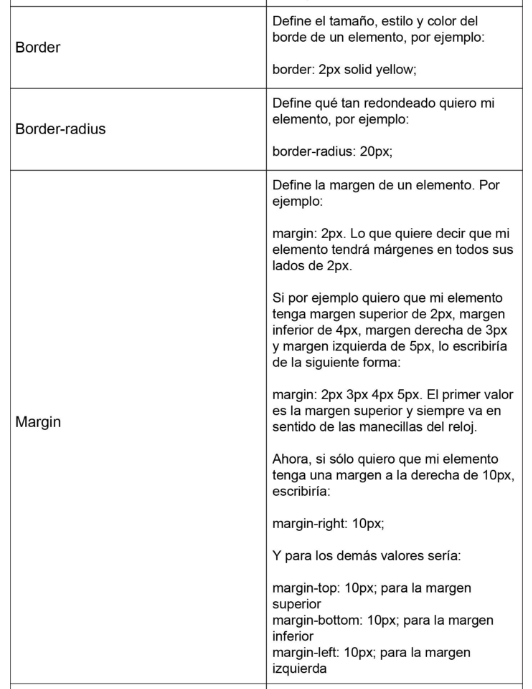


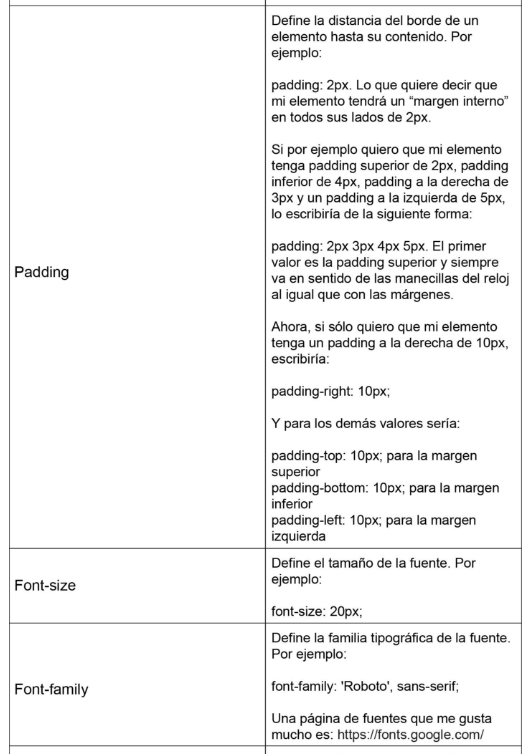
**Displays en CSS** Todos los elementos en CSS son cuadrados o rectángulos y aparte de eso, estos elementos tienen un comportamiento que se define a través de la propiedad display. Los display más comunes y usados son: block, inline, inline-block, none, table, flex y grid. Veamos de qué se tratan:

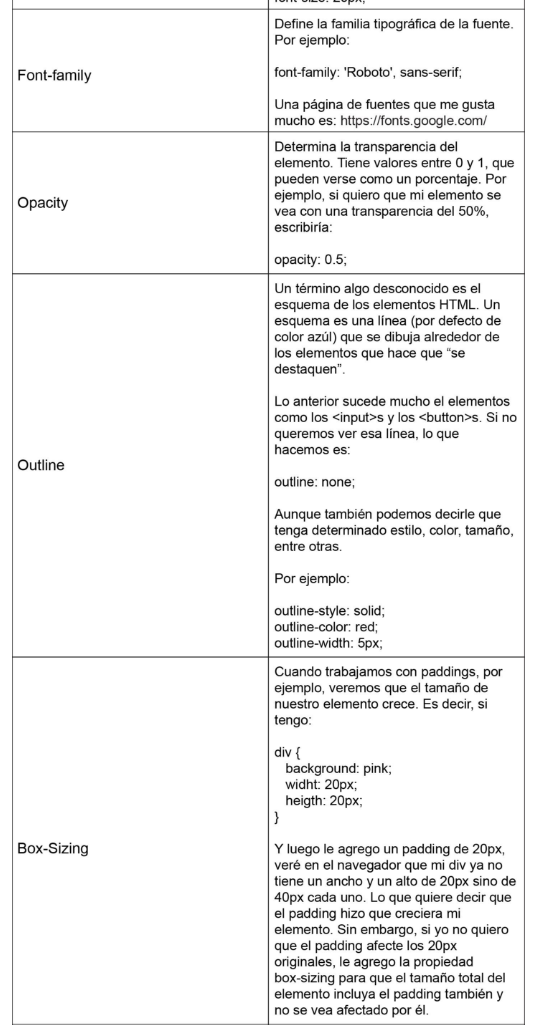


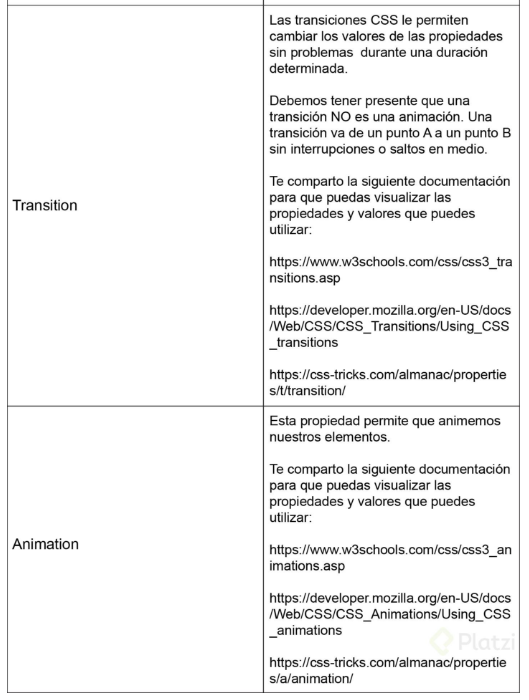
# Funciones de las propiedades CSS más usadas











# Posicionamiento en CSS

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

En CSS los elementos se posicionan utilizando las propiedades top (superior), bottom (inferior), left (izquierda) y right (derecha), pero sólo funcionarán si la propiedad position está establecida. Esto quiere decir que si quiero que mi elemento div esté completamente a la derecha, debo escribir en mi CSS lo siguiente:

div { position: absolute: right: 0px; }

La propiedad **position** tiene 7 valores diferentes: relative, absolute, fixed, sticky, static, initial e inherit. Veremos de qué se tratan:



# *Arquitectura CSS*

# ¿Qué son y para qué nos sirven las arquitecturas CSS?

# Los objetivos son:

# Predecible > Escribir reglas claras.

# 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 prácticas

# Establecer reglas para que el equipo se entienda.

# 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

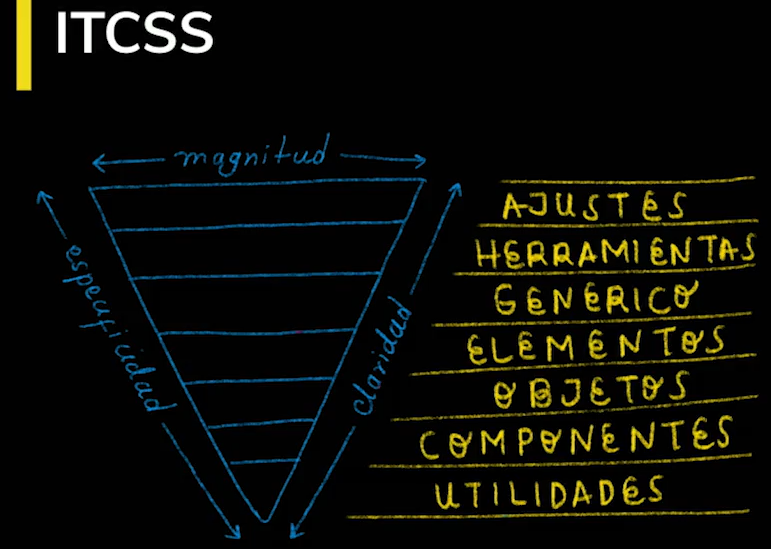
# 

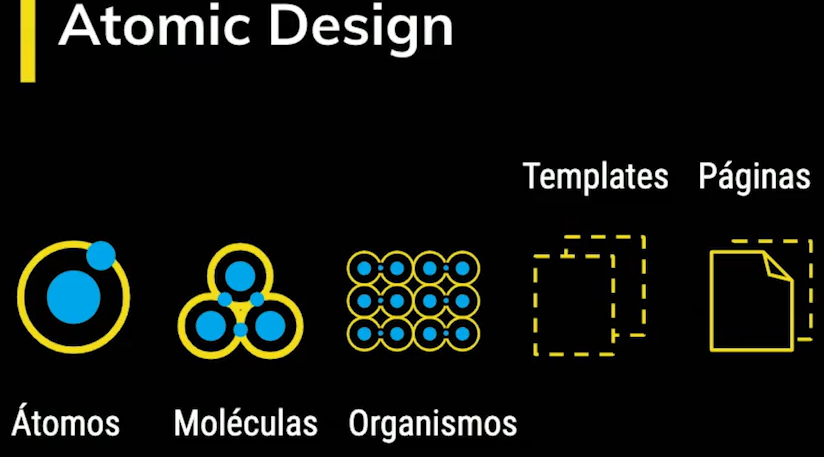
# 

# 

# 

# 





RETO 2 (BEM)

