<https://en.bem.info/methodology/key-concepts/>

<https://cssgradient.io/gradient-backgrounds/>

<https://cssreference.io/>

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

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

<https://htmlreference.io/>

<https://www.freecodecamp.org/settings>

https://necolas.github.io/normalize.css/8.0.1/normalize.css

Internet: Interconnected y Network  
Red de computadoras conectadas en todo el mundo.

Tim Berners-Lee Invento World Wide Web.  
Fundo el consorcio W3C para estandarizar el desarrollo de las tecnologías base Internet

* HTTP Hyper Text Transfer Protocol 😕/ (Permite la comunicación de datos entre dispositivos)
* URL Uniform Resource Locator: (Dirección de un sitio web)
* HTML Hyper Text Markup Language (Lenguaje para describir la estructura de una sitio web)

En 1994 apareció CSS Cascade Style Sheets.

**2. HTML y CSS definición y usos**  
**Definiciones**

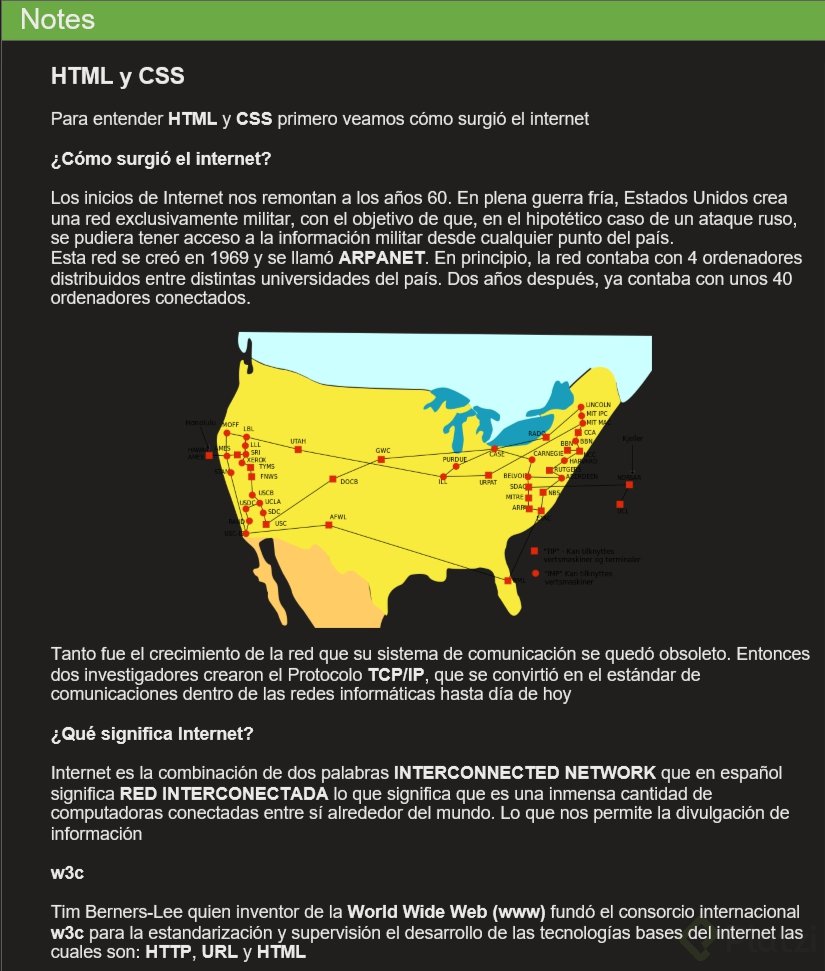
* Internet  
  A principios de los años 80 nació esta tecnología Interconected Network o red interconectada, y esta es la que hoy en día nos permite compartir sonidos, videos e imágenes.
* WWW  
  En 1990 Word Wide Web; fue creada por Tim Berners-Lee el cual también creo el consorcio de la W3C
* W3C  
  Word Wide Web Consortium y es el consorcio que estandariza y supervisa las tecnologías del internet HTTP, URL Y HTML

*- HTTP*  
Hyper Text Transfer Potocol ó protocolo de transferencia de hipertexto es la bases que permite la comunicación entre los dispositivos conectados en la red por ejemplo computadores y servidores

*- URL*  
Uniform Resource Locator ó el localizador de rescursos uniformes más conocido como la dirección de un sitio web, es la manera que agregamos un punto a la red

*- HTML*  
Hyper Text Markup Language ó lenguaje de marcado de hiper text esta es lo que le da la estructura un sitio web como dándonos como herramienta: o lista encabezado, párrafo etc.

* CSS  
  Cascade style sheets ú Hoja de estilo en cascada la cual fue creada en 1994 y son unas series de reglas que describen la apariencia de un sitio web y es el navegador que se encarga de agregar estos estilos a nuestro sitio. esta surgió por la necesidad de mejorar la apariencia de los sitios web, en ella podemos definir el estilo de los componentes html como su tamaño, color, fuente entre otras cosas.



**HTML**: 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**: Es un lenguaje que nos permite crear páginas web con un diseño agradable para los usuarios. Tampoco es un lenguaje de programación.

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

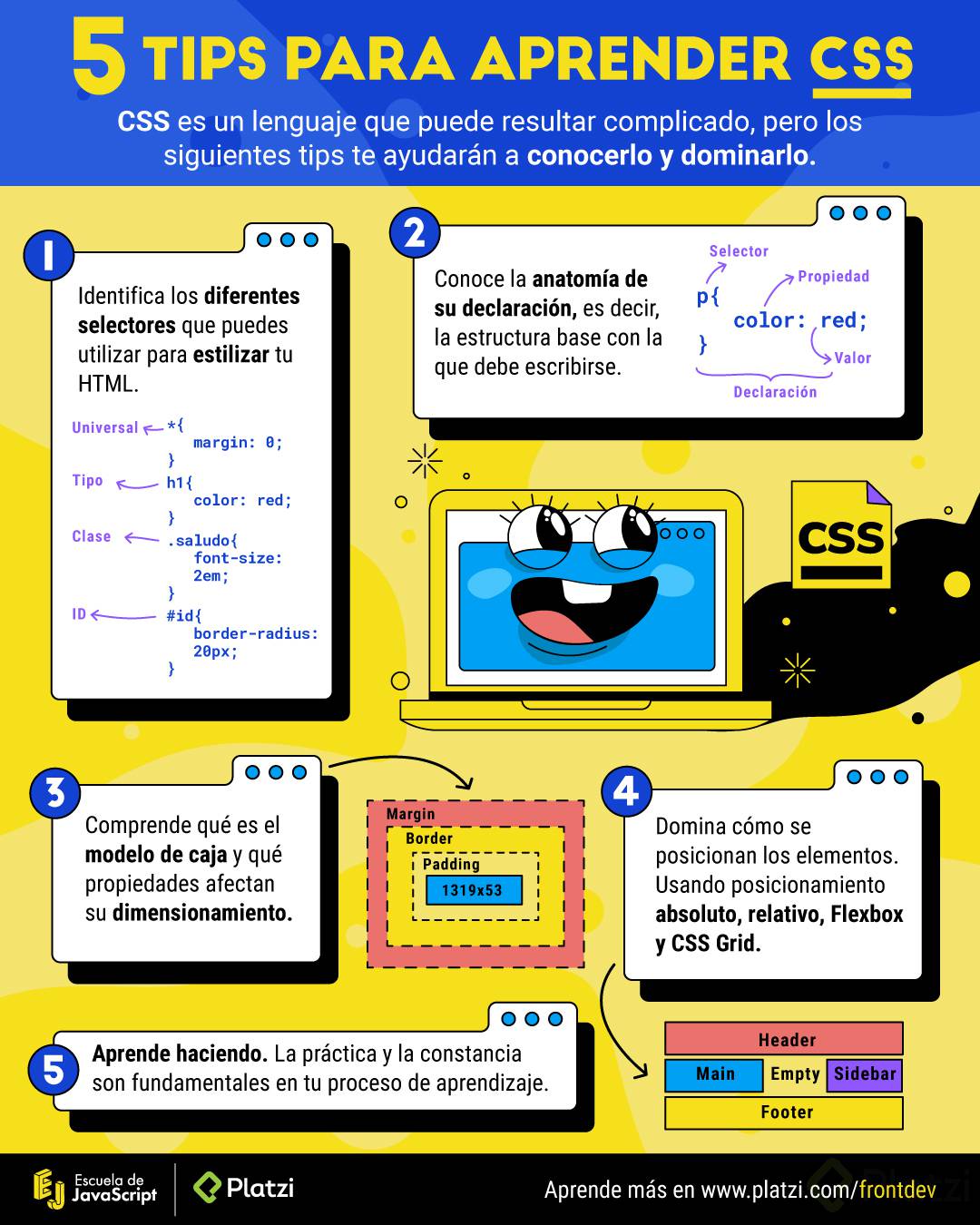
**CSSOM**: así como el DOM para el HTML, EL CSSOM es una representación de objetos de nuestros estilos en CSS.

**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.





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>**.

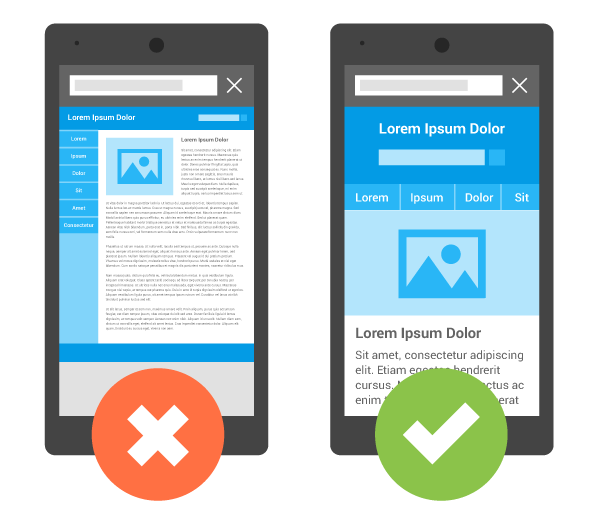
Las etiquetas de apertura también pueden tener atributos. Los atributos nos permiten definir características especiales para nuestros elementos: **<etiqueta atributo=""valor del atributo"">**. Por ejemplo: **<h1 class=""saludo"">**.

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"">**.

Ya que no quedo claro en el video les cuento que la etiqueta meta con el atributo viewport sirve para definir la escala visual del sitio.

En pocas palabras si defines también el atributo content después del atributo viewport en la misma etiqueta meta con el valor: "width=device-width, initial-scale=1" le estamos diciendo a nuestro sitio web que ancho máximo de nuestra página web será el ancho del dispositivo desde donde se esté aperturando el sitio web (móvil, tablet, pc).

Les dejo una pequeña imagen de un sitio web donde se compara el uso de la etiqueta meta con el atributo viewport definido.



**ANATOMÍA DE UN DOCUMENTO HTML DOCTYPE,HTML,HEAD Y BODY:**

<!DOCTYPE html>

Da a conocer la versión de HTML(en este caso la versión 5), lo que nos asegura que el documento html sea analizado de la misma manera en los diferentes navegadores.

**<html>**

Es la etiqueta principal, también conocida como “rute element”. Dentro de esta etiqueta encotramos el <head> y el <body>, los cuáles son hijos de <html>

**<head>**

No tiene ninguna implicación visual en el navegador, pero es muy importante ya que dentro de esta etiqueta nos encontramos la codificación de los caracteres "<meta charset = "UTF-8>.  
Las etiquetas <meta> sirven para añadir características y compatibilidad a la web.

**<title>**

Es el título de la página web, es visible en la pestaña del navegador.

**<body>**

Lo que coloquemos en esta etiqueta será visible directamente en la web. “<h1>”, “<p>”, “<section>”, etc.

<style>

Nos permite añadir estilos sin necesidad de añadir un archivo externo, con la etiqueta:

<link rel="stylesheet" href="style.css">```

Al hacer en nuestro editor de código **html:5** vemos que nos aparece la siguiente estructura:

<html lang="en">

<head>

<meta charset="UTF-8">

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

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

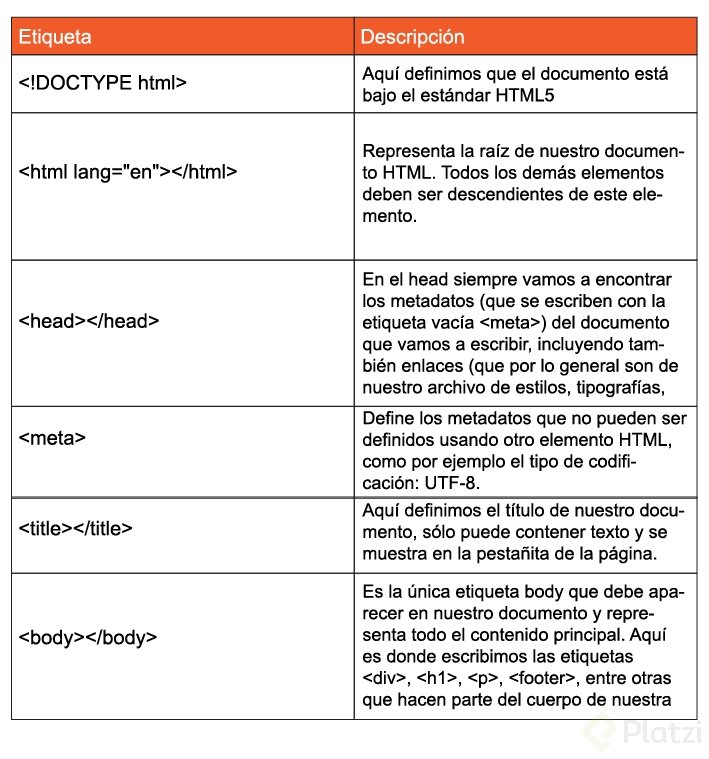
</head>

<body>

</body>

</html>

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:

<html lang="en">

<head>

<meta charset="UTF-8">

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

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>Document</title>

</head>

<style>

/\* Aquí iría nuestro CSS \*/

</style>

<script>

// Aquí debería ir nuestro JavaScript

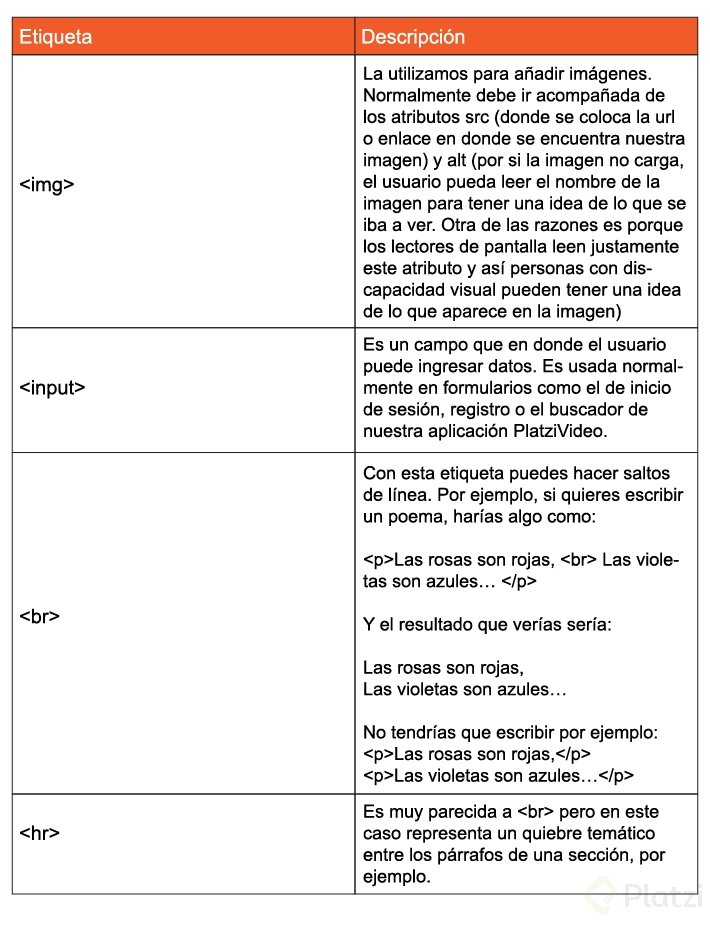
</script>

<body>

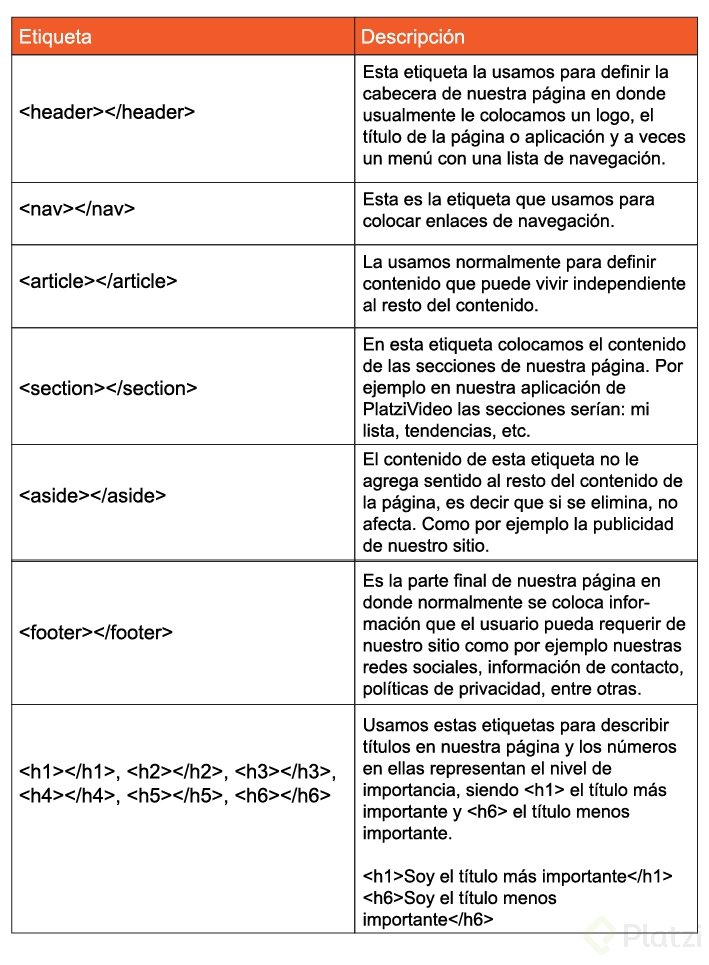
</body>

</html>

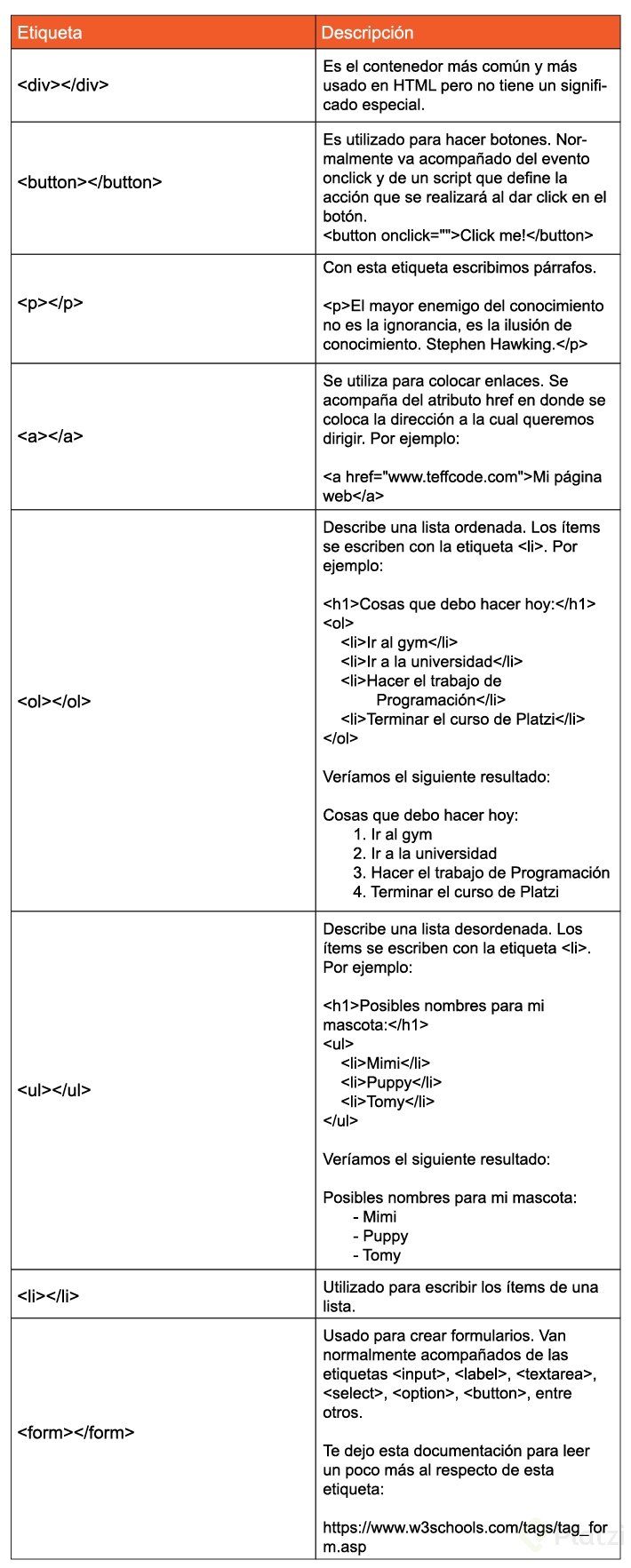
Descripción de algunos elementos vacíos:



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



Descripción de otras etiquetas muy usadas:



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.

**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).

/\* 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 {

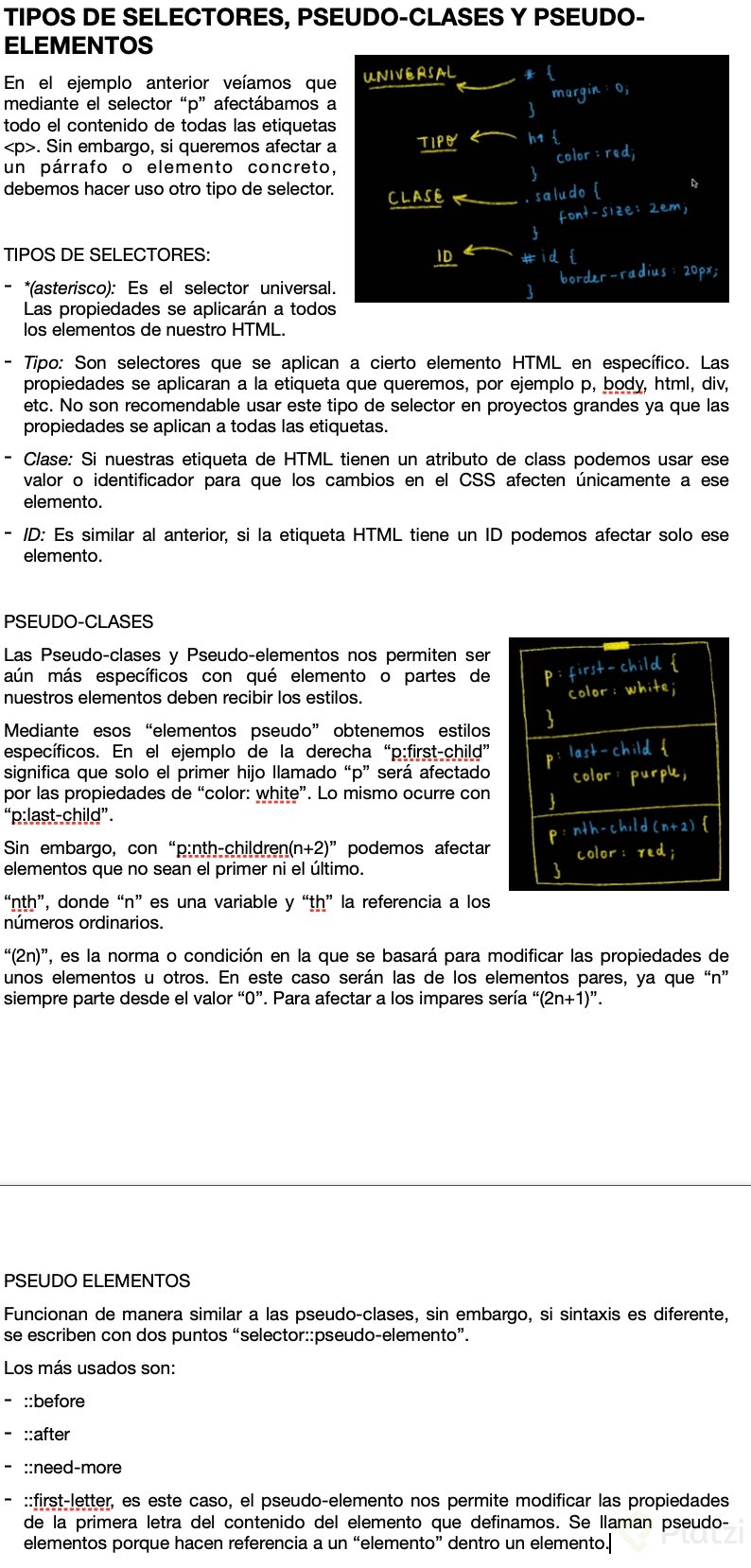
color: purple;

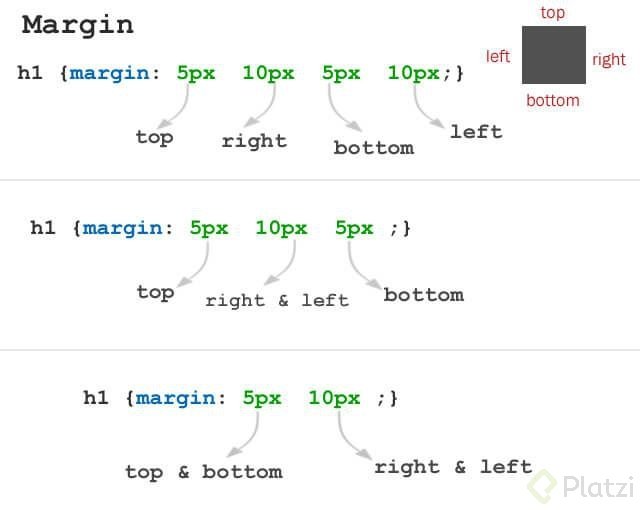
}

**p**:nth-child(2n) {

color: red;

}





**Valores relativos y absolutos**

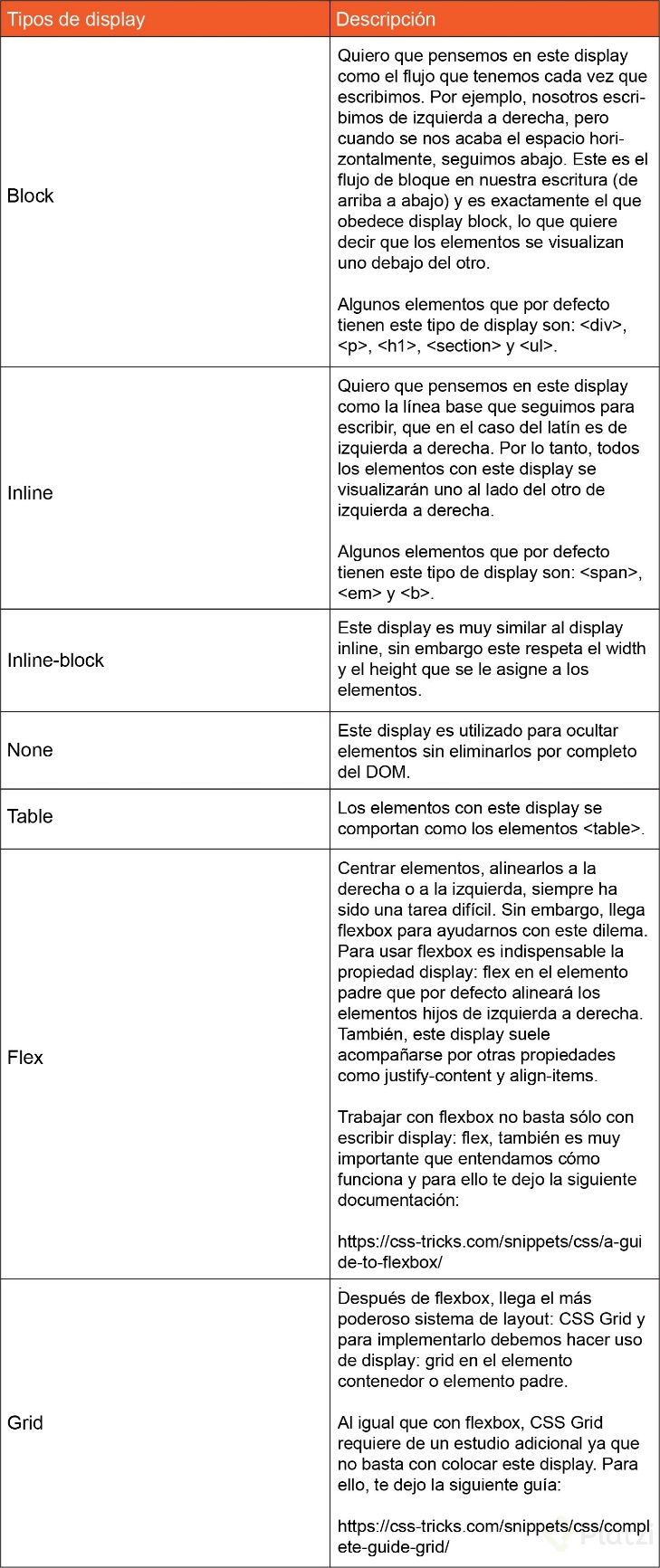
Los valores absolutos son, por ejemplo, centímetros, milímetros, pixeles y pulgadas. Se llaman de esta forma porque no tienen en cuenta a nadie más, no depende de la medida de otra unidad.

Los valores relativas, 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.

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**

**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 que me gusta mucho para sacar colores en hexadecimal es [colorhunt.co](https://colorhunt.co/).
* **RGB**: Donde la R significa Red, G significa Green y B significa Blue; por lo que escribimos rgb(red, green, blue) y 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 la margen de un elemento. Por ejemplo: margin: 2px (lo que quiere decir que mi elemento tendrá márgenes en todos sus lados de 2px).

Si quiero que mi elemento tenga margen superior de 2px, margen inferior de 4px, margen derecha de 3px y margen izquierda de 5px, lo escribiría de la siguiente forma: margin: 2px 3px 4px 5px. El primer valor es la margen superior y siempre va en sentido de las manecillas del reloj.

Si solo quiero que mi elemento tenga una margen a la derecha de 10px, escribiría margin-right: 10px;. Y para los demás valores sería margin-top: 10px; para la margen superior, margin-bottom: 10px; para la margen inferior y margin-left: 10px; para la margen izquierda.

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

Si quiero que mi elemento tenga padding superior de 2px, padding inferior de 4px, padding a la derecha de 3px y un padding a la izquierda de 5px, lo escribiría de la siguiente forma: padding: 2px 3px 4px 5px. El primer valor es la padding superior y siempre va en sentido de las manecillas del reloj al igual que con las márgenes.

Si solo quiero que mi elemento tenga un padding a la derecha de 10px, escribiría: padding-right: 10px;. Y para los demás valores sería padding-top: 10px; para la margen superior, padding-bottom: 10px; para la margen inferior y padding-left: 10px; para la margen izquierda.

**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 que me gusta mucho 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, si quiero que mi elemento se vea con una transparencia del 50%, escribiría: 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 en elementos como los <input>s y los <button>s. Si no queremos ver esa línea, lo que hacemos es outline: none;. Aunque también podemos decirle que tenga determinado estilo, color, tamaño, entre otras. Por ejemplo:

**outline-style**: **solid**;

**outline-color**: **red**;

**outline-width**: 5**px**;

**box-sizing**: Cuando trabajamos con paddings, por ejemplo, veremos que el tamaño de nuestro elemento crece. Es decir, si tengo:

**div** {

background: pink;

width: 20px;

height: 20px;

}

Y luego le agrego un padding de 20px, veré en el navegador que mi div ya no tiene un ancho y un alto de 20px, sino de 40px cada uno. Lo que quiere decir que el padding hizo que creciera mi elemento. Sin embargo, si yo no quiero que el padding afecte los 20px originales, le agrego 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 le permiten cambiar los valores de las propiedades sin problemas durante una duración determinada. Debemos tener presente que una transición NO es una animación. Una transición va de un punto A a un punto B sin interrupciones o saltos en medio.

Te comparto la siguiente documentación para que puedas visualizar las propiedades y valores que puedes utilizar:

* <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 que animemos nuestros elementos.

Te comparto la siguiente documentación para que puedas visualizar las propiedades y valores que puedes utilizar:

* <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/>

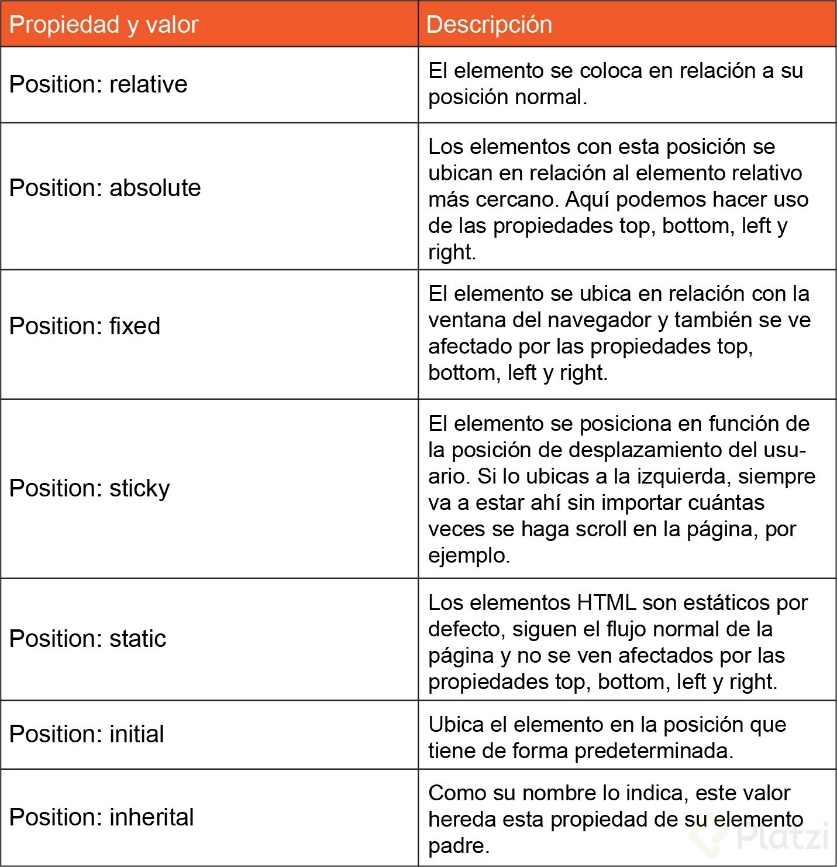
**Posicionamiento en CSS**

El posicionamiento en CSS es una de las cosas más importantes, pues establece cómo van a estar ubicados 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:



Un resumen de la clase.

Los objetivos son:

* **Predecible** > Escribir reglas claras.
* **Reutilizable** > No escribir codigo redundante.
* **Mantenible** > Que sea facil de leer y adaptarnos a los estandares.
* **Escalable** > Que pueda crecer facilmente pero sin afectar el rendimiento.

Buenas practicas

* 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 codigo.

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

[**https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/**](https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/)

[**http://getbem.com/introduction/**](http://getbem.com/introduction/)

[**http://smacss.com/**](http://smacss.com/)

[**https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/**](https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/)

[**https://bradfrost.com/blog/post/atomic-web-design/**](https://bradfrost.com/blog/post/atomic-web-design/)

**¿Qué es un componente? Analicemos nuestros diseños**

En esta clase la profesora Estefany Aguilar nos explica qué es un componente en el mundo del frontend. También analizaremos el diseño del proyecto del curso.

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. Puedes apreciarlos en las plataformas que visitas todos los días: Twitter, Facebook, Platzi, YouTube y muchas más.

Creación de un carousel de imágenes con CSS: Detalle de cada item

El selector ~ de CSS nos permite dar estilos a todos que cumplan los requisitos y sean “hermanos directos”, es decir, que tengan el mismo elemento padre.

Por ejemplo:

.PrimerTitulo ~ **h2** {

background-color: red;

}

<article class="ContenedorPadre">

<h2 class="PrimerTitulo">Este es el primer elemento H2</h2>

<!-- Este título tendrá color de fondo rojo -->

<h2>Este es el segundo elemento H2</h2>

<div class="ElementoPadreDiferente">

<!-- Este título NO tendrá color de fondo rojo porque su padre es diferente -->

<h2>Este es el tercer elemento H2</h2>

</div>

<!-- Este título también tendrá color de fondo rojo -->

<h2>Este es el cuarto elemento H2</h2>

</article>

También existe el selector **+**. Solo aplica los estilos al primer hermano directo de nuestros elementos.  
Por ejemplo:

.PrimerTitulo + **h2** {

background-color: blue;

}

<article class="ContenedorPadre">

<h2 class="PrimerTitulo">Este es el primer elemento H2</h2>

<!-- Este título SÍ tendrá color de fondo azul -->

<h2>Este es el segundo elemento H2</h2>

<div class="ElementoPadreDiferente">

<!-- Este título NO tendrá color de fondo azul -->

<h2>Este es el tercer elemento H2</h2>

</div>

<!-- Este título no tendrá color de fondo azul -->

<h2>Este es el cuarto elemento H2</h2>

</article>

**Flexbox**

Uno de los trabajos más difíciles en CSS es alinear elementos. Para hacerlo más fácil podemos contar con Flexbox.

Es importante tener presente que tendremos un contenedor y los elementos que queremos organizar dependiendo de nuestras necesidades.

<https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

<https://flexboxfroggy.com/#es>

<https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox>

https://platzi.com/blog/flexbox-vs-css-grid-cual-es-la-diferencia/

**Nuestro nuevo sistema de layout: CSS Grid**

Con CSS Grid podemos maquetar todo el layout/estructura general de nuestro sitio para que se adapten a diferentes tamaños de pantalla, lo que conocemos como diseño responsivo.

Al igual que Flebox, tenemos propiedades diferentes, tanto para el contenedor como para los elementos, y podemos usarlos dependiendo de nuestras necesidades.

<https://css-tricks.com/snippets/css/complete-guide-grid/>

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

<https://labs.jensimmons.com/>

<https://www.youtube.com/channel/UC7TizprGknbDalbHplROtag>

<https://cssgridgarden.com/#es>

En esta clase vamos a continuar con la construcción de nuestro proyecto PlatziVideo. Para esto vamos a reutilizar la estructura y estilos del menú desplegable que construimos en las siguientes clases:

* [Estructura con HTML y BEM de un menú desplegable](https://platzi.com/clases/1640-frontend-developer/21890-estructura-con-html-y-bem-de-un-menu-desplegab-3/)
* [Estilizando nuestro menú desplegable con CSS](https://platzi.com/clases/1640-frontend-developer/21891-estilizando-nuestro-menu-desplegable-con-c-5/)

**Media queries**

Los media queries nos ayudan a ajustar nuestros diseño a dispositivos más pequeños como tablets y celulares.

Es recomendable seguir la metodología de Mobile First: comenzar a diseñar para el dispositivo más pequeño e ir creando las media queries para las pantallas más grandes.

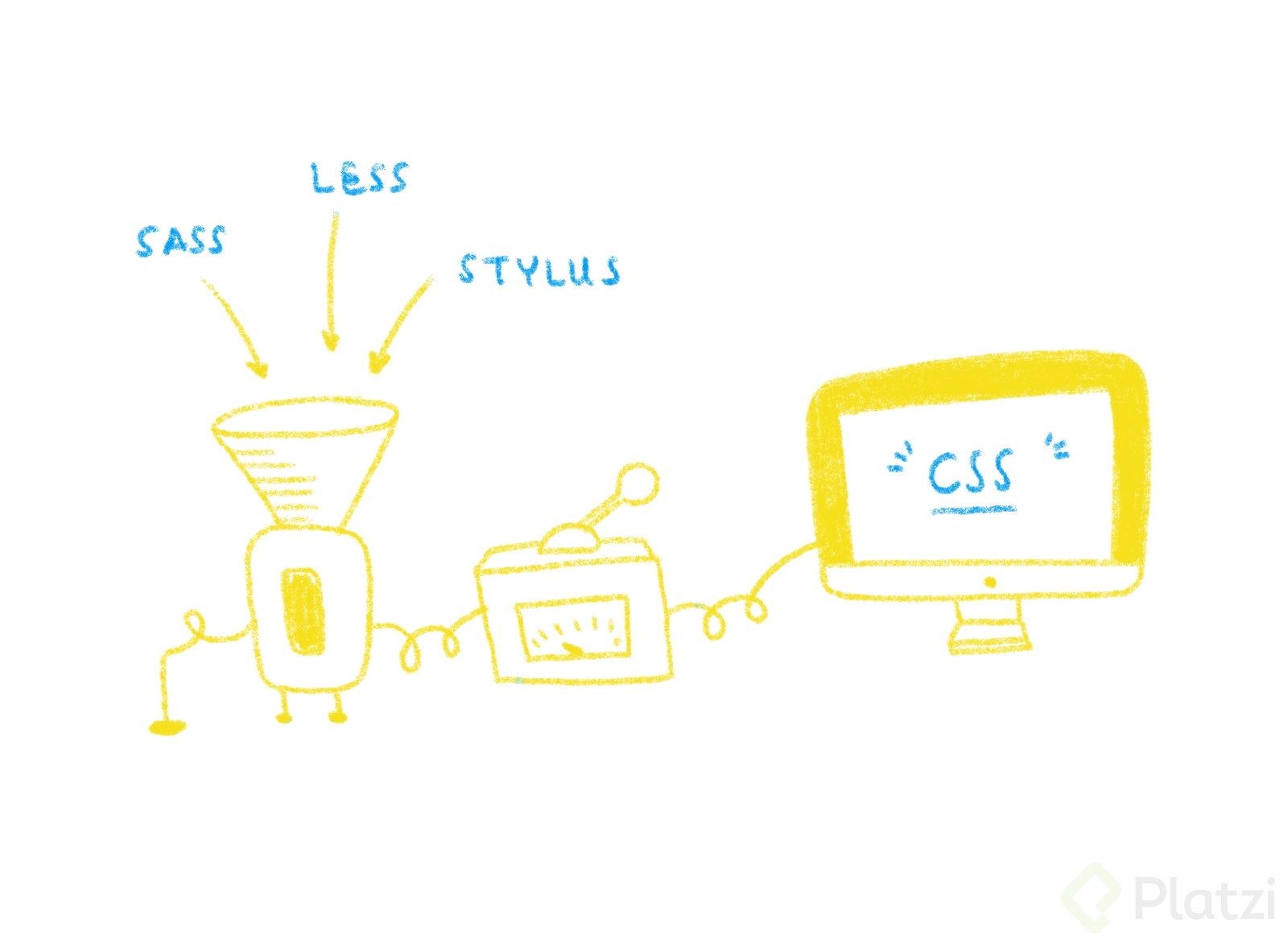
**¿Qué es un preprocesador, cuáles existen y cuáles son sus diferencias?**

CSS es un lenguaje de hojas de estilo que nos permite crear sitios web agradables para el usuario, sin embargo, nuestros archivos de CSS suelen ser bastante extensos, lo que produce una demanda significativa de nuestro tiempo y puede generar un trabajo menos productivo.

Una de las cosas que puedes hacer para evitar tantas líneas de código es utilizar los preprocesadores de CSS, los cuales extienden las funcionalidades de CSS común, permitiéndonos tener variables, funciones, mixins, reutilización de código, flexibilidad en el desarrollo, etc.

Pero, ¿cómo es que funcionan los preprocesadores?

Un preprocesador se escribe con una sintaxis especial que nosotros le indicamos y debe compilarse a CSS para ser comprendido por el navegador. En sí lo que estamos haciendo es CSS pero con esteroides.



Esta sintaxis que te menciono depende de cada preprocesador. Los más conocidos y usados son: LESS, SASS y Stylus. ¿Cuál usar? En mi opinión personal, esta decisión depende más de tus gustos personales y de qué tan cómodo o cómoda te sientas con una sintaxis o con otra, sin embargo, es también importante que dialogues con tu equipo y evalúen con qué preprocesador quieren trabajar dependiendo de las necesidades del proyecto.

Aquí te comparto la documentación oficial de cada uno de ellos para que puedas evaluar sus diferencias en sintaxis:

<https://sass-lang.com/guide>  
<http://lesscss.org/>  
<http://stylus-lang.com/>

Para aprender muy bien CSS y este tema de preprocesadores, es muy importante que practiques y seas constante. Comenzar algo siempre va a ser muy difícil, pero con paciencia y dedicación podrías llegar a dominar estos temas fácilmente.

Algo que me ayudó mucho cuando comencé, fue replicar páginas que me gustaban mucho (a modo de estudio, por supuesto). Comencé haciendo extensas líneas de CSS y quizás mis creaciones no eran las más bonitas, pero no me desmotivé, al contrario, seguí intentándolo y sumándole a esta práctica el tema de los preprocesadores, JavaScript, librerías, frameworks, etc.

Así que ánimo, practica bastante y por qué no, comparte tu conocimiento con la comunidad de Platzi o comunidades de desarrollo del lugar en donde vives.

**La accesibilidad y nuestra responsabilidad como desarrolladores**

Debemos pensar en esas personas con una discapacidad visual que no tienen la posibilidad de ver lo mismo que la mayoría de nosotros. Estas personas no siempre usan el mouse, sino lectores de pantalla.

Un Lector de Pantalla se encarga de leer toda la aplicación elemento por elemento. Que los lectores de pantalla funcionen es responsabilidad de las y los desarrolladores: debemos tener muy buena semántica, usar las etiquetas y atributos adecuados entre otras.

Programa para accesibilidad: ANDI - NVDA – JAWS -VOICEOVER

Para los que se hayan confundido o quieran saber la diferencia entre extend y el include es la siguiente:  
Los Mixins de Sass, hacen algo parecido a lo que hemos visto con Extend. Encapsulan varias líneas de código que posteriormente podemos llamar dentro de un selector.  
Sin embargo, los Mixins incorporan una novedad. Permiten pasarles un argumento que podrá tener un valor distinto cada vez que lo usemos.  
Me explico. Los Mixins se crean de la siguiente manera:

@mixin border-radius($radius) {

-webkit-border-radius: $radius;

-moz-border-radius: $radius;

-ms-border-radius: $radius;

border-radius: $radius;

}

.box { @include border-radius(10px); }

Fuente: [https://www.espai.es/blog/2018/04/sass-mixins-extend/](https://platzi.com/clases/1640-frontend-developer/21906-hablemos-de-variables-herencia-anidamiento-operado/url)

Variables=> $(nombrevariable): característica; ejempl. $blanco: White. /// color: $blanco;

Anidamiento (anida las clases)=> .clase {“elemento a afectar (clase, id, elemento)”{características del estilo}} // funciona cuando la clase padre hace parte de los elementos hijos

Heredar => se utiliza con una clase: .flex {display: flex} y en la clase que estoy trabajando ejemplo .header, coloco: @extend .flex

Reutilización de código: @mixin “nombre clase” { características de estilos} para usarlo se coloca @include “nombre clase”

Storybook: guía para documentar los componentes de una aplicación. Para que sea mas sencillo ver los componentes y/o comportamientos en acción. Para correr=> en la terminal : npm run storybook

