**Clase1. ¿Qué son y** para **qué nos sirven HTML y CSS?**

**HTML** es un lenguaje de marcado usado para decirle a tu navegador cómo estructurar las páginas que visitamos. Este no es un lenguaje de programación.

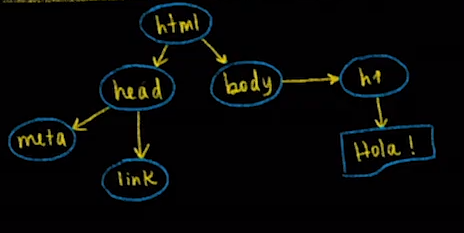
**CSS** es un lenguaje que nos permite crear páginas web con un diseño agradable para los usuarios. Tampoco es un lenguaje de programación.

**Clase 2. 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.

El navegador realiza 5 pasos para poder entender el código que hemos escrito en HTML:

1. Convierte el código en **bytes** para poder entenderlo.
2. Convierte en caracteres dependiendo de la codificación que usemos (Por ejemplo: UTF-8)
3. Convierte los caracteres en tokens para ordenarlos, automáticamente detecta a que etiqueta hace relación la agrupación de caracteres (W3C)
4. Hace transformación a nodos los objetos.
5. Organiza los objetos en el DOM, sigue la estructura de árbol, donde su raíz es la etiqueta principal **<HTML>** y desde allí empieza a bifurcarse de manera jerárquica.

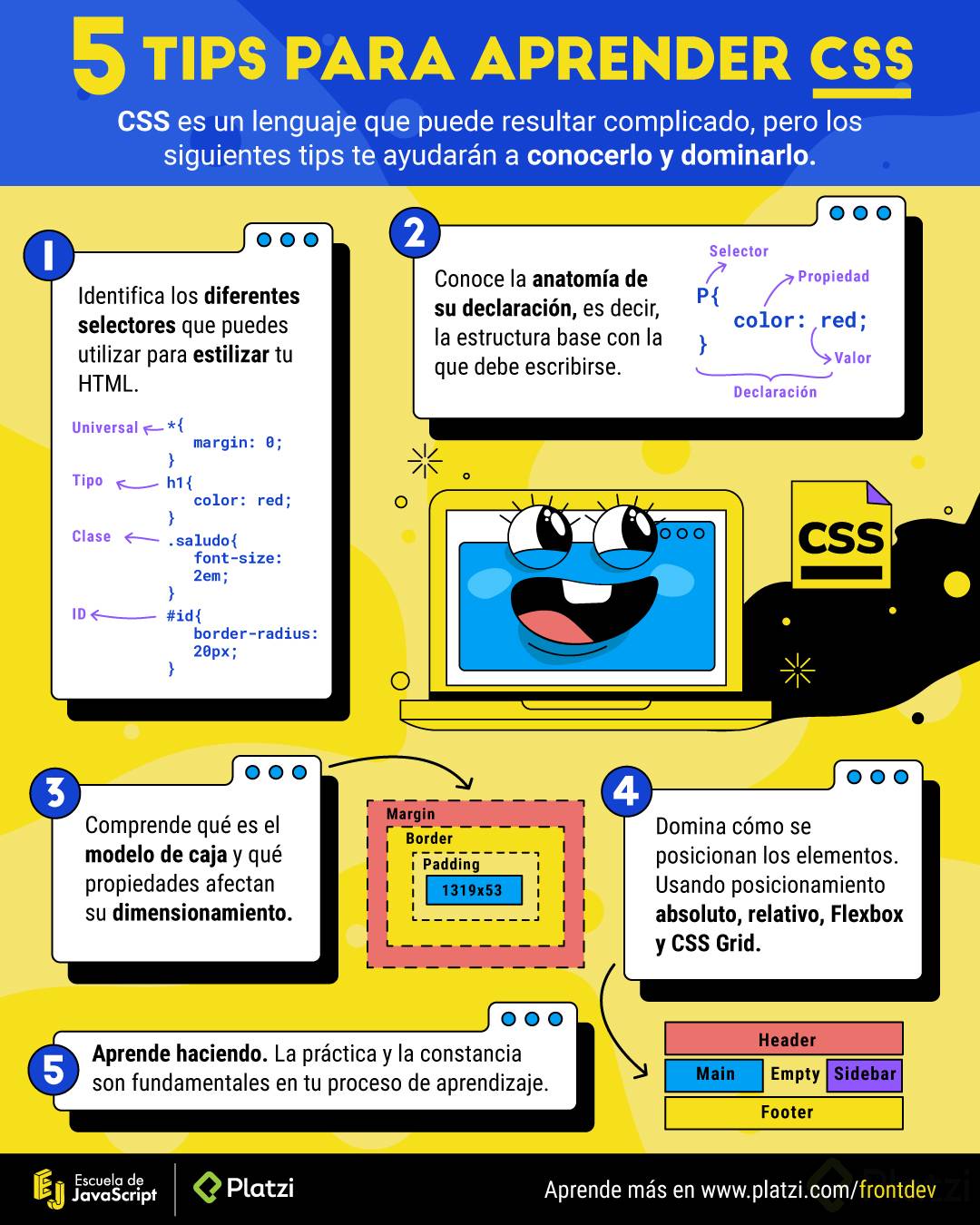


**CSSOM**: Así como el DOM para el HTML, EL CSSOM es una representación de objetos de nuestros estilos en CSS. El CSS hace una transformación pero se pega a la transformación hecha por el DOM, asignándole a cada objeto de este el CSS que le corresponde.

**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.



**Clase 3. 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"">.

**Clase 4. Anatomía de un Documento HTML: DOCTYPE, html, head y body**

* **DOCTYPE:** Nos asegura que el documento sea analizado de la misma manera en los diferentes navegadores.
* **HTML:** Es la etiqueta principal que también es conocida como **root element**.
* **HEAD:** No tiene ninguna implicación visual dentro del navegador. Dentro se declara la decodificación en caracteres. Además, se les declara otros tipos de meta. Una de sus propiedades extras es el título.
* **BODY:** Aquí estará el cuerpo o esqueleto de HTML.

Dentro del archivo **HTML** se puede agregar una zona para el CSS utilizando una etiqueta **style**, una buena práctica es crear un archivo aparte, exclusivo con el contenido CSS y consumirlo desde el HTML.

**Clase 5. 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.

**Clase 6. 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.