**Frontend.**

**¿Qué es Frontend?**

Frontend es la parte de un programa o dispositivo a la que un usuario puede acceder directamente. Son todas las tecnologías de diseño y desarrollo web que corren en el navegador y que se encargan de la interactividad con los usuarios.

Un programador Frontend debe saber de códigos (HTML, CSS y JavaScript) para poder usar algunos framework o librerías que expanden sus capacidades para crear cualquier tipo de interfaces de usuarios. React, Redux, Angular, Bootstrap, Foundation, LESS, Sass, Stylus y PostCSS son algunos de ellos.

# Backend.

**¿Qué es Backend?**

Es lo opuesto a un **Front-end**, ya que el Back-end trabaja en el lado del *servidor* mientras el Front-end en el lado del *cliente*.

Un programador Backend es el responsable de manejar toda la lógica que hay detrás de una petición dada por el navegador.  
Una característica que lo diferencia del **Front-end** es que no tiene estándares, ya que tiene varios lenguajes de programación **(Node.js, Python, PHP, Ruby, GO, Java, .NET entre otros)** . Los cuales estos lenguajes tienen sus propios frameworks como Django (Python), Lavarel (PHP), Rails (Ruby), Express (JS), Spring (Java).

# Fullstack

Un Desarrollador **Fullstack** es alguien que tiene conocimientos tanto del **Front-end** como del **Back-end**, es alguien que entiende cómo funciona un producto web desde principio a fin. Desde la ***idea***, hasta la ***producción***.  
Sin dudas tiene un alto conocimiento acerca de una aplicación web.  
Pero Hey! Ellos tienen sus áreas de especialización, ya sea **Front-end** o **Back-end**

**Full Stack**: Entenderlo todo. NO saberlo todo.

**Internet.**

Internet: **Inter**connected y **Net**work  
Red de computadoras conectadas en todo el mundo.

Tim Berners-Lee inventó **W**orld **W**ide **W**eb.  
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.

**Referencias de HTML y CSS.**

<https://cssreference.io/>  
<https://htmlreference.io/>

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

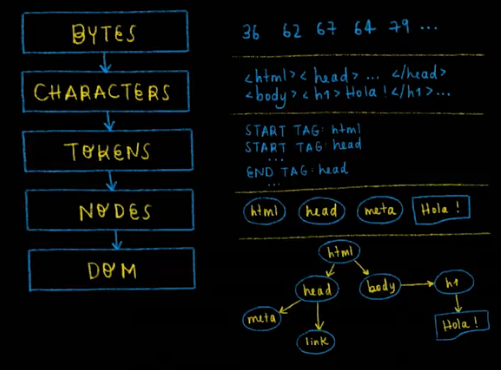
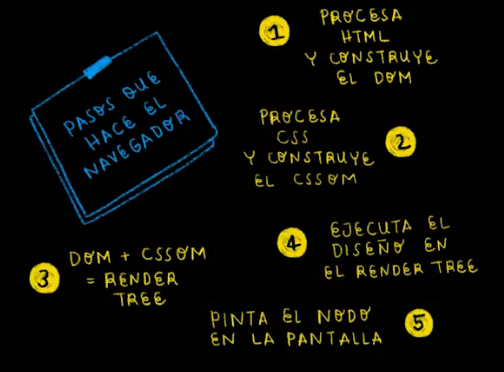
**DOM**: Document Object Model. Es una transformación del código HTML escrito por nosotros a objetos entendibles para el navegador.

**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.

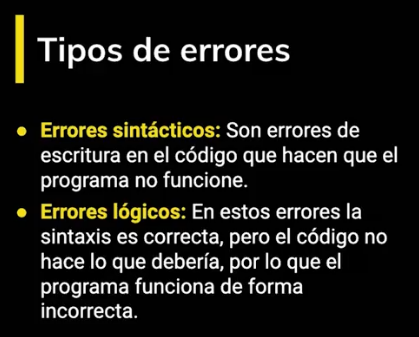
# 5 tips para aprender CSS.

# 

# 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.



**LINK para validar problemas de Accesibilidad.**

<https://www.ssa.gov/accessibility/andi/help/install.html>

**LINK para validar nuestro HTML.**

[https://validator.w3.org](https://platzi.com/clases/1640-frontend-developer/21880-tipos-de-errores-en-html-debugging-y-servicio-de-v/url)

**LINK para elegir paleta de colores.**

<https://colorhunt.co/>

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

**¿Cómo se construye la tecnología web?**

El **Frontend** es la parte del software que interactúa con los usuarios, en cambio, el **Backend** es la parte que no puedes ver, esta oculta porque contiene la información privada o sensible de nuestros usuarios.

HTML nos permite crear la estructura de la página: títulos, párrafos, menús, etc. CSS nos permite configurar los estilos del HTML, los colores, formas, posiciones, etc. Además, JavaScript *(JS)* se encarga de la parte funcional, nos permite conectarnos con otros servicios (como PayPal) y nos ayuda a hacer las interacciones mucho más fluidas.

Los **Compiladores** nos ayudan a construir el frontend de nuestras páginas web de forma mucho más rápida y sencilla. En vez de escribir el mismo código una y otra vez, vamos a utilizar estas herramientas para obtener estos mismos resultados de forma automatizada.

El trabajo y las operaciones del backend se pueden resumir con las siglas **CRUD**: *Create*, *Read*, *Update* y *Delete*. Cuando el frontend se comunica con el backend, debe indicar el tipo de operación y los datos necesarios para que todo funcione correctamente.

Por ejemplo: cuando registramos un usuario, el frontend debe mandar los nombres y contraseña de este usuario para que, el backend, pueda guardar esta información en la base de datos y podamos consultarla más adelante.

# Comprendamos Internet

Internet es un conjunto descentralizado de redes de comunicación interconectadas, en otras palabras, son dos (o más) computadoras que se conectan entre sí. Los Protocolos son un conjunto de reglas que hacen posible la comunicación entre diferentes elementos que forman parte de un sistema.

La **World Wide Web** es un sistema de distribución de documentos (de hipertexto o hipermedia) interconectados y accesibles vía internet, mientras que, los **Hipertextos** son textos que contienen enlaces a otros textos.

**FTP** es el protocolo de transferencia de archivos entre sistemas conectados a una red, así es cómo diferentes personas podemos compartir documentos entre nosotros.

Hay algunas computadoras que solo utilizamos para entregar documentos los conocemos como **servidores**, en cambio, las computadoras que solo leen y reciben estos documentos los conocemos como **clientes**. También existe el protocolo de comunicación **P2P** (Peer to Peer) donde una misma computadora trabaja como servidor y cliente al mismo tiempo.

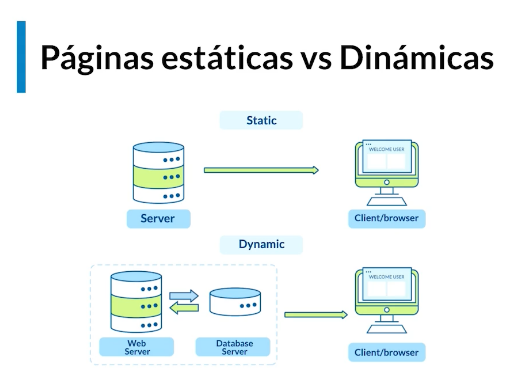
La **tecnología de la información** es la aplicación de ordenadores y equipos de telecomunicación para almacenar, recuperar, transmitir y manipular datos, con frecuencia, utilizado en el contexto de negocios o empresas.

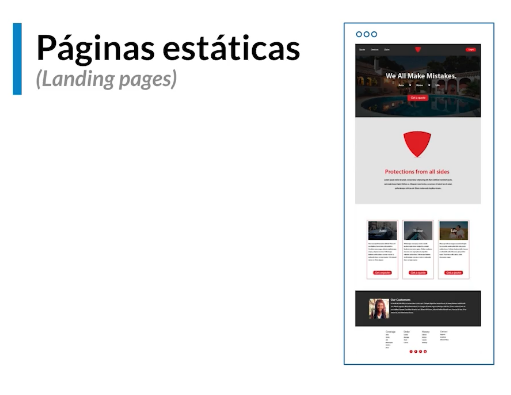
La **Comunicación Síncrona** es comunicación en tiempo real, por ejemplo, en aplicaciones de mensajería y videochat. Sin embargo, también existe la **Comunicación Asíncrona**, comunicación en tiempo NO real, así funciona Gmail, puede pasar que recibamos un nuevo correo, pero no lo leemos inmediatamente lo envían, por eso es comunicación asíncrona.

***Entidades HTML.***

Empiezan con “&” y terminan con “;”(semicolon). Son caracteres reservados que nos permiten inyectar caracteres especiales.

[***https://dev.w3.org/html5/html-author/charref***](https://dev.w3.org/html5/html-author/charref)



# Anatomía de una etiqueta de HTML

 **Etiquetas y sus atributos**

Las **Etiquetas** son fragmentos de texto rodeados por corchetes angulares (< y >) con funciones y usos específicos:

<html>Contenido</html>

Los **Atributos** afectan a los elementos por su presencia o enriquecen la definición de la misma. Por ejemplo, el atributo lang en la etiqueta html indica que el contenido de esta página está en un idioma específico:

<html lang="es">Contenido en Español</html>

Las etiquetas **meta** (*meta tags*) son etiquetas que se incorporan en el encabezado de una página web y son invisibles para los usuarios normales, sin embargo, son de gran utilidad para navegadores u otros programas que pueden valerse de esta información:

<head>

<meta name="description" content="Descripción de nuestra página">

</head>

También encontramos etiquetas comunes como <footer> para agrupar el contenido del pie de página, <span> para guardar texto genérico y sin reglas de espaciado o tamaño predeterminadas, <img src="imagen-url.png"> para incluir imágenes, entre otras.

**Listado de las etiquetas más usadas**

En este listado se encontrarán algunas de las etiquetas más utilizadas y para qué sirven.

<!doctype html>

Define que el documento está bajo el estándar de HTML.

<head>

Representa una colección de metadatos acerca del documento, incluyendo enlaces a, o definiciones de, scripts y hojas de estilo.

<title>

Define el título del documento, el cual se muestra en la barra de título del navegador o en las pestañas de página. Solamente puede contener texto y cualquier otra etiqueta contenida no será interpretada.

<base>

Define la URL base para las URLs relativas en la página.

<link>

Usada para enlazar JavaScript y CSS externos con el documento HTML actual.

<meta>

Define los metadatos que no pueden ser definidos usando otro elemento HTML.

<style>

Etiqueta de estilo usada para escribir CSS en línea.

<body>

Representa el contenido principal de un documento HTML. Solo hay un elemento <body> en un documento.

<section>

Define una sección en un documento.

<nav>

Define una sección que solamente contiene enlaces de navegación

<article>

Define contenido autónomo que podría existir independientemente del resto del contenido.

<aside>

Define algunos contenidos vagamente relacionados con el resto del contenido de la página. Si es removido, el contenido restante seguirá teniendo sentido

<h1>, <h2>, <h3>, <h4>, <h5>, <h6>

Los elemento de cabecera implementan seis niveles de cabeceras de documentos; <h1> es la de mayor jerarquía y <h6> es la de menor importancia. Un elemento de cabecera describe brevemente el tema de la sección que introduce.

<header>

Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos.

<footer>

Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación.

<address>

Define una sección que contiene información de contacto.

<main>

Define el contenido principal o importante en el documento. Solamente existe un elemento <main> en el documento.

<form>

Representa un formulario, con atributos de controles que puede ser enviado a un servidor para procesamiento.

<img>

Representa una imagen.

<iframe>

Representa un contexto anidado de navegación, es decir, un documento HTML embebido.

<a>

Representa un hiperenlace, enlazando a otro recurso.

<em>

Representa un texto enfatizado, como un acento de intensidad.

<strong>

Representa un texto especialmente importante .

<small>

Representa un comentario aparte, es decir, textos secundarios como un descargo de responsabilidad o una nota de derechos de autoría, que no son esenciales para la comprensión del documento.

Fuente: <https://developer.mozilla.org/es/docs/HTML/HTML5/HTML5_lista_elementos>

# Estructura básica de HTML y head

# 

***Links. <link>***

Permite referenciar otros archivos para ser ligados al proyecto.

<link rel="stylesheet" href="css/styles.css">

***Orden List. (<ol>)***

Para realizar listas con numeraciones usamos <ol> dentro marcamos cada Item con <li> (List Item).

            <ol> <!-- Listas Ordenadas -->

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

            </ol>

***Un-orden List. (<ul>)***

Para realizar listas con bullets usamos <ul> dentro marcamos cada Item con <li> (List Item).

            <ul> <!-- Listas desordenadas -->

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

                <li>

                    <!-- Item -->

                </li>

            </ul>

***Anclas (<a>)***

<a href="#">Photo</a>

<a href="#">Imagenes</a>

<a href="#">Icon</a>

***Images.<img>***

Nos permite posicionar una imagen. Source (src) es donde se colocará la ruta de la imagen. Alternative (alt) es el texto alternativo que mostrará en caso de que la imagen no sea cargada.

<img src="" alt="">

***Etiquetas Contenedoras.***

<header>

</header>

<nav>

</nav>

<main>

</main>

<section

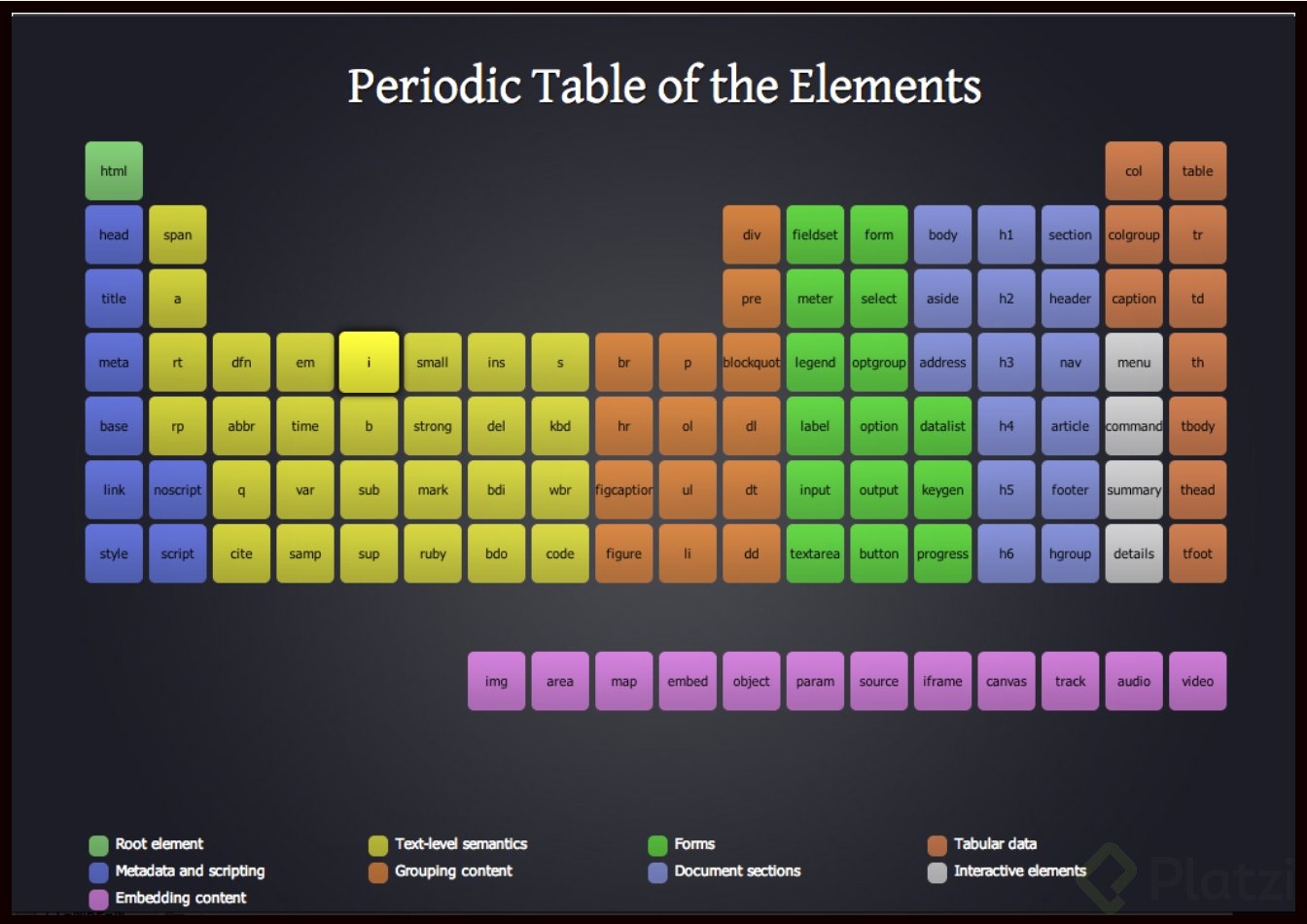
</section>

<div>

</div>

<footer>

</footer>



**Composición básica de HTML.**

**<!Doctype HTML> Indica que es un archivo HTML**

**<HTML lang=”en”> Nuestra etiqueta HTML principal / Marcamos el idioma**

**<HEAD>**

**Aquí se colocarán los links ,etiquetas <meta> u otras configuraciones generales**

**</HEAD>**

**<BODY>**

**Aquí colocaremos todo nuestro contenido <header> <main><section><footer>**

**</BODY>**

**</HTML>**

# Nombrando etiquetas para CSS según nuestra estructura en HTML

Para estructurar nuestro proyecto, necesitamos ordenar nuestra definición de etiquetas con CSS. La conexión entre nuestra estructura HTML y CSS da como resultado una página web. A continuación, veremos las reglas de nomenclatura que dan orden a nuestra definición de CSS.

Para darle nombre a los estilos podemos utilizar diferentes métodos, aunque es recomendable seguir estos pasos:

-No utilizar palabras en español: la mayoría de los lenguajes no aceptan ñ o tildes, es por eso que evitamos esta característica.

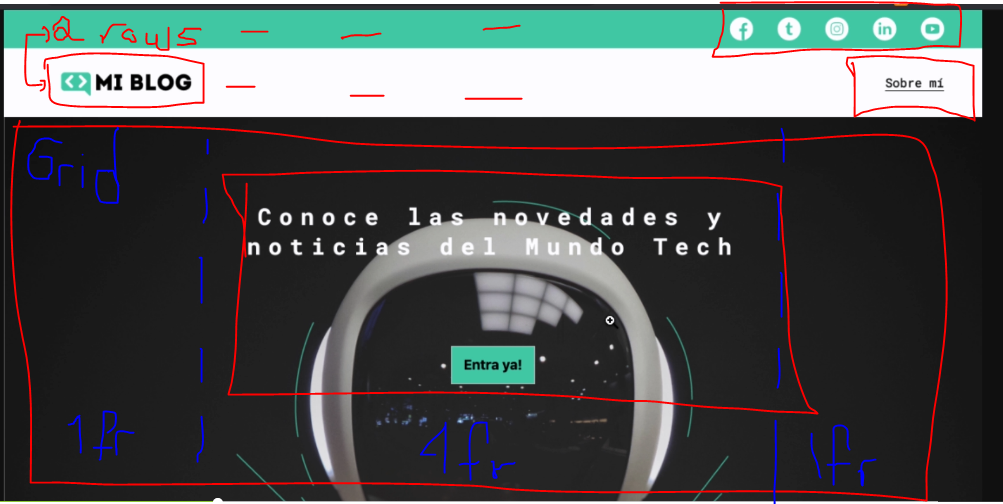
-Para los espacios utilizamos guión medio o guión bajo, recuerda usar solo uno de los dos caracteres.

-Describir el nombre de manera general a particular, por ejemplo: home\_bar

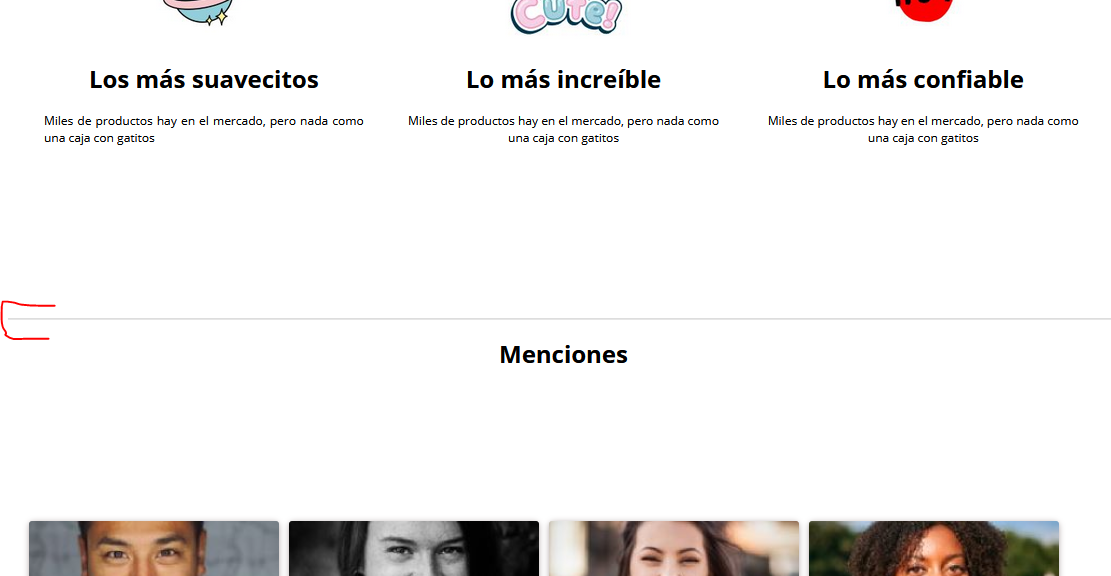
También podemos definir las siguientes etiquetas, las cuales nos permiten crear secciones dentro de HTML y evitamos un concepto llamado ***divismo***.

**<section>** Lo usamos para dividir secciones  
**<footer>** Define el footer ya sea de toda la página o solo de un componente.  
**<iframe>** Nos ayuda a incrustar otra página en nuestro html  
**<video>** Nos ayuda a incrustar video dentro de nuestra web

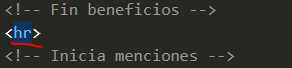
**Ejemplo de Maquetación Visual.**



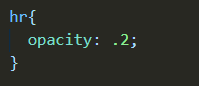
***Generar línea Divisora.***



*HTML: Se agrega la etiqueta <hr>*



*CSS:*



**Tipos de imágenes**

# 

# 

# 

# Optimización de imágenes.

# 

# 

<https://tinypng.com/>

[*https://www.verexif.com/*](https://www.verexif.com/)

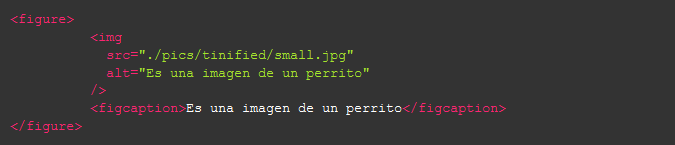
# Etiqueta img

La etiqueta imagen nos ayuda a poder renderizar las imágenes, tiene 2 atributos, el primero es **src=’’ (**source**)** para decirle en donde se encuentra la imagen que queremos enseñar.

El segundo atributo es **alt=’’** que es una descripción que le vamos a dar a la imagen y esto nos sirve para dos cosas: primero, cuando la imagen no se logre renderizar va poder decir ahí cual es la descripción de esa imagen en caso de que no se vea, el segundo es alt que nos ayuda para temas de accesibilidad.



La etiqueta de imagen no es la única que tenemos, hay otra que se llama figure, esto nos ayuda para poder generar un contenedor para la imagen.



**Paginas para descargar imágenes gratis para proyectos:**

[*https://www.pexels.com/es-es/*](https://www.pexels.com/es-es/)

[*http://styleguides.io/*](http://styleguides.io/)

<https://www.freepik.es/>

<https://icons8.com/illustrations>

<https://unsplash.com/>

# Imágenes responsive

# Recordar usar la etiqueta <picture></picture>

# Y también utilizar la etiqueta <source/> dentro de esta, a la vez que se ordenan de mayor tamaño a menor con el argumento media, al final dejar la imagen con la etiqueta <img> que esta imagen debe ser la pensada para dispositivos móviles.

<body>

    <main>

      <!--

            Para trabajar con imagenes responsivas se recomienda usar la etiqueta picture.

            y dentro de esta indicar los diferentes tamaños dentro de las etiquetas source y el atributo media.

            Ademas de que el orden es de mayor a menor tamaño y al final la imagen con la etiqueta img que esta pensada para mobile.

         -->

      <picture>

        <source media="(min-width: 1000px)" srcset="./images/large.jpg" />

        <source media="(min-width: 800px)" srcset="./images/medium.jpg" />

        <img src="./images/small.jpg" alt="image" />

      </picture>

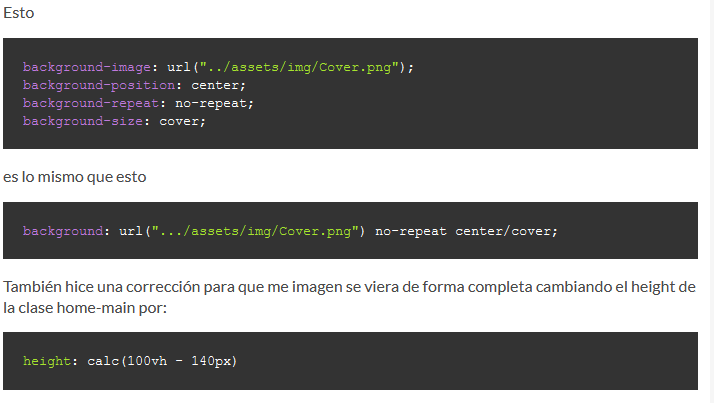
    </main>

  </body>

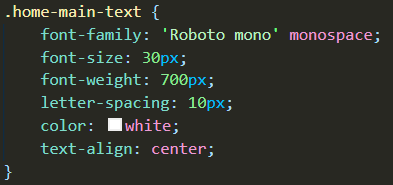
Es completamente válido usar **<picture>** dentro de **<figure>** y así poder usar el **<figcaption>**

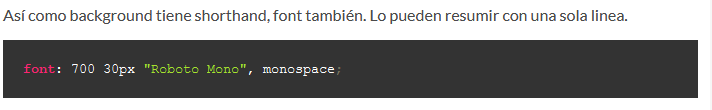


**Para acomodar imágenes de Background. Shorthand background.**



**Shorthand de Font.**





# Fuentes de iconos

Todo menú necesita de la presencia de iconos, por este motivo, en esta clase aprenderás sobre las fuentes de iconos y las aplicarás en la realización del menú de tu proyecto.

En este abordaje a las fuentes de iconos, conocerás la plataforma [icomoon.io](http://icomoon.io), en ella podrás importar o añadir familias de iconos, desde tu computador o desde el sistema.

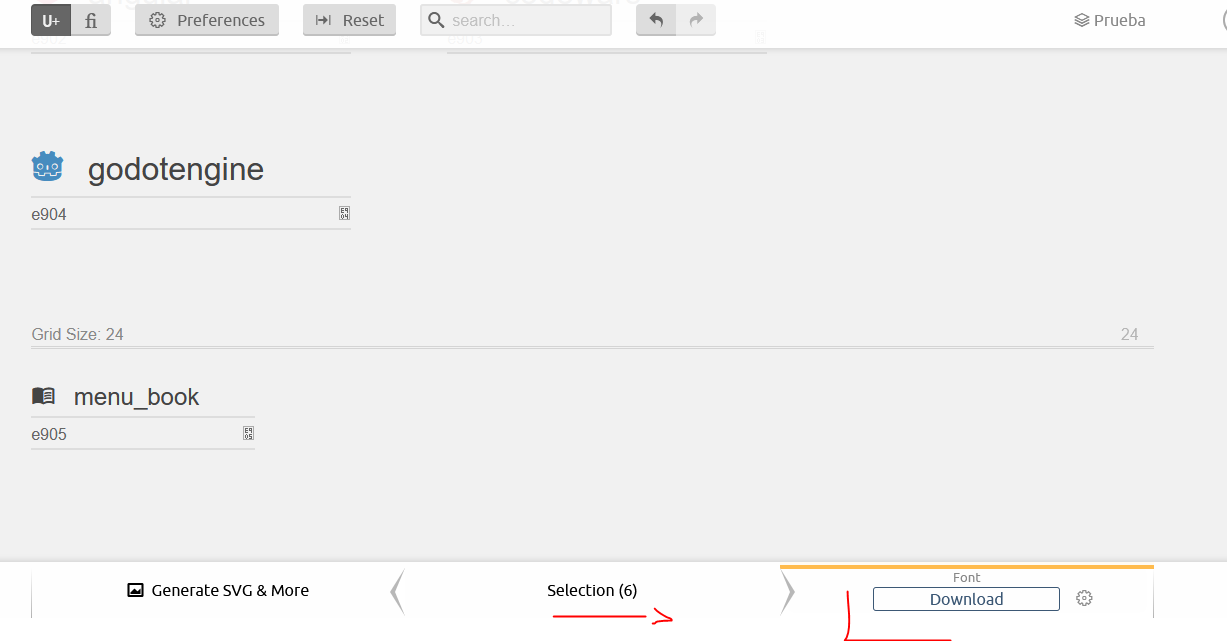
[*https://www.flaticon.es/*](https://www.flaticon.es/)

*https://svgporn.com/*

[*https://iconscout.com/*](https://iconscout.com/)

[*https://icomoon.io/*](https://icomoon.io/)





Al descargar los archivos encontraremos un archivo *style.css* de código con la etiqueta @font- face, que acogerá a otros atributos como font- family, font- style y font- weigth, font- variant, que serán necesarios en nuestro proyecto.

* Esto deberemos pasarlo a nuestros estilos.

@font-face {

  font-family: 'icomoon';

  src:  url('fonts/icomoon.eot?pni8ak');

  src:  url('fonts/icomoon.eot?pni8ak#iefix') format('embedded-opentype'),

    url('fonts/icomoon.ttf?pni8ak') format('truetype'),

    url('fonts/icomoon.woff?pni8ak') format('woff'),

    url('fonts/icomoon.svg?pni8ak#icomoon') format('svg');

  font-weight: normal;

  font-style: normal;

  font-display: block;

}

[class^="icon-"], [class\*=" icon-"] {

  /\* use !important to prevent issues with browser extensions that change fonts \*/

  font-family: 'icomoon' !important;

  speak: never;

  font-style: normal;

  font-weight: normal;

  font-variant: normal;

  text-transform: none;

  line-height: 1;

  /\* Better Font Rendering =========== \*/

  -webkit-font-smoothing: antialiased;

  -moz-osx-font-smoothing: grayscale;

}

.icon-anchor:before {

  content: "\e900";

  color: #8940fa;

}

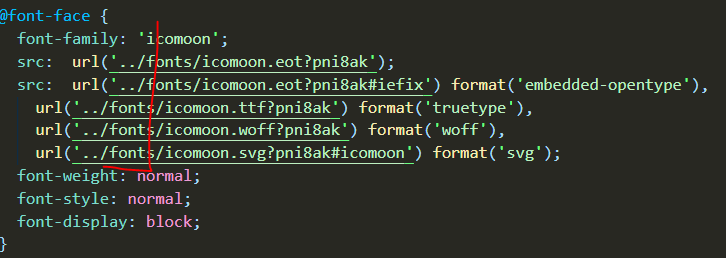
.icon-android:before {

  content: "\e901";

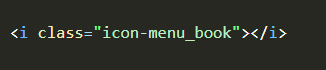
  color: #3ddc84;

}

* También tendremos que copiar la carpeta *fonts* a nuestro proyecto.
* Luego cambiaremos las rutas que estaban colocadas en el css anteriormente copiado, por las rutas apuntando a nuestro proyecto.



* Para agregarlo al proyecto solo debo llamar la clase en mi elemento HTML.



# Etiqueta video.

<!- Controls "nos da los botones" \*\*\*\* preload "permite que el video se auto cargue antes de que el usuario lo manipule" -->

        <video controls preload="auto">

          <!--Para no tener problemas en los navegadores

         defino varias rutas que yo tenga del video en distintos formatos-->

          <source src="./miVideo.mp4#t=10,60" />

          <source src="./miVideo.mpv#t=10,60" />

          <!-- Para darle un tiempo de inicio a fin lo hacemos con el # después t="como quiero que inicie" lo separo con , despues le doy el otro tiempo donde quiero que

          finalize-->

        </video>

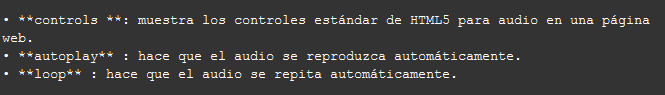
# Videos HTML5

Como sabes, los videos son contenidos cada vez más comunes e incluso necesarios en las web sites. Por esta razón, en esta clase aprenderás a incluir un video en tu proyecto y, por lo tanto, lo modificarás para convertirlo en un material responsivo, es decir, que sea coherente con tu trabajo de Responsive Design.

Para aprender esto es necesario que elijas cualquier video que tengas en su formato original y aplicar las siguientes líneas:

**HTML**

    <video class="html-video" src="videos/f8.mp4" width="1280" height="720px" controls></video>



**CSS**:

.html-video {

  width: 100%;

  height: auto;

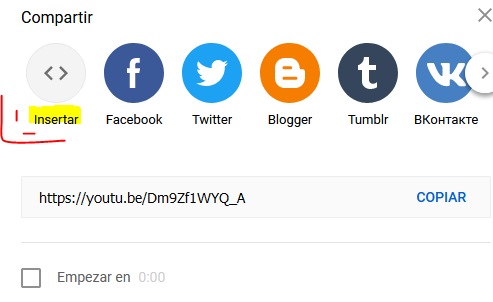
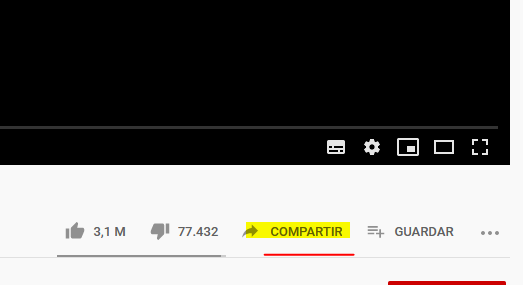
}

# Video insertado

En esta clase aprenderás a implementar videos responsive en tu proyecto, originarios o alojados en otras plataformas (como YouTube y/o Vimeo). Por este motivo trabajarás con la etiqueta iframe.

Ésta etiqueta hará posible que el video que insertes tenga la posibilidad de adaptarse a distintas formas de visualización, aunque, a pesar de ellas, aprenderás cuáles son las diferencias entre usar un video inserto y hacerlo desde html5.

Para extraer un video de YouTube:



La etiqueta <iframe></iframe> se utiliza para insertar contenido de otras páginas web dentro de la tuya, como por ejemplo videos de YouTube.

<div class="flexible-video">

    <iframe class="youtube-video" width="560" height="315" src="https://www.youtube.com/embed/Dm9Zf1Q\_A"

      frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture"

      allowfullscreen>

    </iframe>

  </div>

A diferencia de los videos HTML (tema anterior), en este caso los tamaños de width y height no los puede calcular automáticamente.

Entonces NO sirve poner:

.youtube-video {

  width: 100%;

  height: auto;

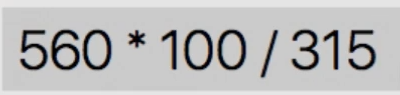
}

Por lo tanto, para mantener las proporciones del video se utiliza un <div> contenedor. En este div se pone sin altura y se utiliza padding-top que, a diferencia de height, calcula el porcentaje en base al tamaño del width. Para saber el valor del padding-top es necesario realizar una regla de 3 simple para calcular los porcentajes adecuados. Para saber la altura se debe hacer: height \* 100 / width.

Y si necesitan saber el porcentaje necesario del ancho es: width \* 100 / height.  
Para que el video quede adentro del div contenedor se utiliza el position: Absolute, como vimos en las otras clases.





  
En conclusión, si el video es de 16:9 en horizontal el código CSS del div contenedor va a quedar:

.flexible-video {

  width: 100%;

  height: 0;

  padding-top: 56.25%;

  position: relative;

}

.youtube-video {

  width: 100%;

  height: auto;

  position: absolute;

  top: 0;

  bottom: 0;

  right: 0;

  left: 0;

  width: 100%;

  height: 100%;

}

# Formularios.

# Etiqueta form e input.

# Según la especificación de HTML se debe trabajar a label como un contenedor.

# Es importante utilizar la etiqueta <form></form> para indicar que utilizaremos un formulario, además de la semántica y buenas prácticas.

# Dentro de estas etiquetas indicaremos los elementos del Formulario como lo son:

# <label></label> que será como nuestro “Contenedor” en esta ocasión. Dentro de él pondremos la etiqueta <span></span> para colocar un texto haciendo alusión al contenido esperado (nombre, contraseña, fecha etc…) .Otra de las etiquetas importantes es <input> Aquí es importante aclarar que hay muchos tipos de Input, los cuales podrán ser consultados acá: <https://www.w3schools.com/html/html_form_input_types.asp>

# También es importante mencionar que el valor del atributo for <label for=" nombre "> debe ser el mismo que el de id <input id=" nombre ">

<body>

  <form action="">

    <label for="nombre">

      <span>Cual es tu nombre?</span>

      <input type="text" id="nombre" placeholder="Tu nombre" required>

    </label>

    <label for="inicio-platzi">

      <span>Que día comenzaste en Platzi?</span>

      <input type="date" id="inicio-platzi" >

    </label>

    <label for="horario">

      <span>En que horario estudias?</span>

      <input type="time" id="horario" >

    </label>

  </form>

</body>

</html>

# Calendar

Para generar un formulario de calendario podemos realizarlo de distintas formas. La primera va a ser una de las más largas ya que vamos a especificar un Input por cada tipo de dato que necesitemos (dia,hora,semana,mes).

Es importante mencionar que el valor de mi atributo **name <input name=" nombre ">** será el nombre con el que se identifique el valor ingresado por el input en los método post.

<form action="">

    <label for="hora">

        <span>Hora</span>

        <input type="time" id="hora" name="hora" />

      </label>

      <label for="dia">

        <span>Día</span>

        <input type="date" id="dia" name="dia" />

      </label>

      <label for="semana">

        <span>Semana</span>

        <input type="week" id="semana" name="semana" />

      </label>

      <label for="mes">

        <span>Mes</span>

        <input type="month" id="mes" name="mes" />

      </label>

</form>

**<input name=" mes ">**



Otra forma de realizarlo esto es:

<form action="">

    <label for="calendario">

      <span>Calendario</span>

      <input type="datetime-local" name="calendario" id="calendario" />

    </label>

    <input type="submit" />

</form>

# Autocomplete y required.

# Para hacer que los campos del formulario se autocompleten solos con datos que ya ha usado antes el usuario (y están guardados en el navegador), se usa el atributo *autocomplete* dentro de la etiqueta input.

# 

Al colocar el atributo ***Required***, el submit no funcionará si el campo indicado no está completado.



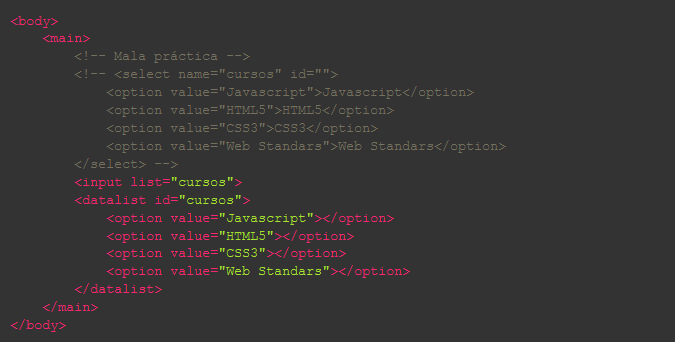
# Input type submit vs. Button tag

# 

# Select.

Para hacer listas de selección, las mejores prácticas es permitir que el usuario inicie a escribir y el desplegable se corresponda con las primeras letras de lo que está escribiendo el usuario.

Para ello, se usa la etiqueta input con el atributo list="nombreID". Luego, la etiqueta datalist con el atributo id="nombreID" y dentro la etiqueta option con el atributo value="nombreValorListaDesplegable" y dejar su contenido vacío para que permita que el usuario sea el que rellene y en base a eso le salga la lista.



# Atributos especiales de las etiquetas para mejorar el funcionamiento de nuestros formularios

Los formularios de nuestras páginas web se componen principalmente de labels para indicarle a los usuarios qué datos van a llenar, los inputs son los espacios donde pueden llenar estos datos y los botones son para enviar el formulario (o cualquier otra cosa, por ejemplo, hacer reset).

¿Te has preguntado por qué usamos labels y no cualquier otra etiqueta? ¿Que tienen de especial los labels?

Los labels no son como otras etiquetas que solo guardan y muestran el texto con los diferentes estilos que definamos en el CSS. Los labels están especialmente preparados y optimizados para que nuestros formularios funcionen correctamente.

## **Atributos para asociar labels a sus respectivos inputs**

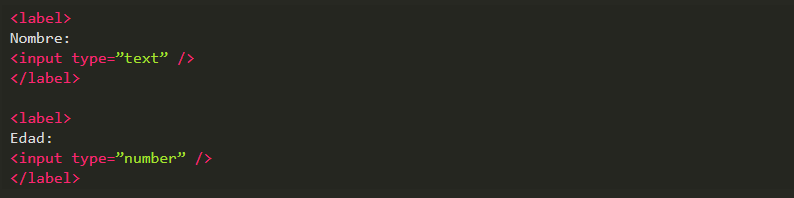
Seguramente has llenado miles de formularios desde tu computadora sin problemas. Pero la probabilidad de que tengas un mal rato cuando te encuentras con un formulario desde tu celular es muy alta. Por ser dispositivos mucho más pequeños es mucho más difícil atinarle al input correcto para empezar a escribir los datos que nos pide el formulario.

De hecho, suele pasar que, debido a todos estos problemas, terminamos abandonando esa página web y nos olvidamos de ella para siempre. Nuestra tarea es evitar este tipo de problemas a toda costa.

Solucionar esto es muy fácil. Vamos a crear la funcionalidad de que, al tocar el label que corresponde a cada input, la página reaccione como si hubiéramos hecho click directamente en el input.

Hay dos formas de hacerlo:

**Primera forma**: Podemos encerrar nuestros inputs dentro de sus respectivos labels:



**Segunda forma**: Sólo debemos añadir un par de atributos a los elementos de nuestro formulario:

* Añadimos el atributo id en la etiqueta input y le damos un valor, recuerda que el valor de nuestros IDs deben ser únicos e irrepetibles para evitar problemas en nuestra página web.
* Añadimos el atributo for en la etiqueta label con el valor del id del input que queremos enlazar.

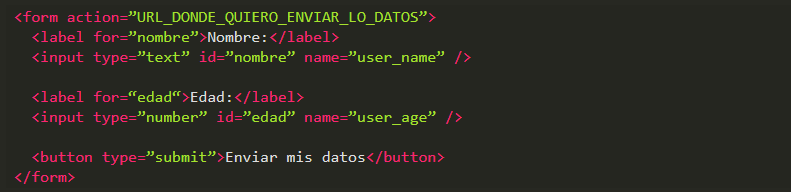


## **Atributos para enviar los datos de nuestros formularios**

El objetivo de los formularios es enviar y almacenar los datos de nuestros usuarios en algún sitio, ¿verdad?

La forma de hacerlo es mucho más fácil de lo que crees. Solo debemos añadir algunos atributos en la etiqueta <form> y en los inputs y botones de nuestro formulario.

* **Action**: Este atributo de la etiqueta <form> nos permite establecer el lugar o URL donde los datos de los usuarios deben ser enviados una vez ellos terminen de llenar el formulario.
* **Name**: Este atributo de las etiquetas <input> nos permite que los datos que los usuarios hayan escrito se envíen correctamente cuando han terminado de llenar el formulario. El valor de esta etiqueta debe indicar el nombre con el cual queremos almacenar o interactuar con esta información en el futuro.
* **Type**: El atributo type=”submit” le permite a los usuarios indicar que han terminado de llenar el formulario y lo pueden enviar para almacenar la información. También podemos usar este atributo en inputs para que estos pasen a tener el rol de botón de envío del formulario.



Si pruebas este mismo formulario vas a ver que, al enviarlo, la página se dirige a la URL que indicaste en el atributo action de la etiqueta form. Pero es no es todo, fíjate bien en la URL y veras que los datos de tus usuarios están escondidos por ahí:



¡Si tienes este mismo resultado significa que has tenido éxito!

Recuerda que si quieres almacenar estos datos en algún lugar como una base de datos puedes aprender un poco de Backend, APIs REST, CRUD y hacer todas las configuraciones necesarias.

# ¿Cómo funciona CSS?

El **CSS** son las hojas de estilo en cascada que definen la apariencia de nuestros documentos en HTML.

Para que nuestros estilos CSS se apliquen correctamente a nuestras páginas web, debemos utilizar la etiqueta link con el atributo href y la ruta a nuestro archivo .css:

<html>

<head>

<link rel="stylesheet" href="estilos.css">

</head>

<body>

... etc ... etc....

</body>

</html>

Los **Selectores** nos permiten conectar las etiquetas de HTML con sus respectivos estilos en CSS.

Existen muchos tipos de selectores, por ejemplo, los selectores de **clase** buscan el contenido que tenga un cierto valor en su atributo class:

CSS (con punto antes del nombre de la clase):

.caja {

color: red;

}

HTML:

<div class="caja">Contenido</div>

También tenemos selectores de tipo **ID** (estos selectores solo pueden aplicar a un elemento, no va a funcionar si escribimos dos o más etiquetas con el mismo ID):

CSS (con `#` antes del nombre del ID):

#caja { color: red; }

HTML:

<div id="caja">Contenido</div>

En CSS utilizamos atributos para definir los estilos de nuestros elementos, podemos modificar el color de la letra, tamaño, color de fondo, espaciado, entre otras:

.caja {

color: red;

background: yellow;

font-size: 10px;

padding: 20px;

}

***/\* Reglas***

La regla es todo el bloque de código que contiene los estilos de una etiqueta en CSS

***/\*Selectores***

El selector es el nombre de esa etiqueta, como ejemplo puede ser (header, body, etc)

\* **Selector de Etiquetas**

body{}, header{}, section{}, nav{}, etc

\* **Selector Descendiente**

body header div nav ol li a {} Se accede al elemento con las etiquetas en cascada

\***Selector de Clase(.)** Se puede usar para múltiples elementos.

Se coloca en el HTML

<div class="juegos"> </div>

Y en el CSS se llama el selector con un punto .

.juegos{color:black;}

\***Selector de ID(#)** Se puede usar solo para un único elemento.

Se llama en el CSS el elemento por el ID con un numeral #

<div id="juegos"> </div>

#juegos{color:black;}

***/\*Declaraciones.***

Es cada línea de comando que modifica el estilo:

background-color: gray;

***/\*Propiedades.***

Como su nombre lo dice es la propiedad de la declaración (background, color , width, etc)

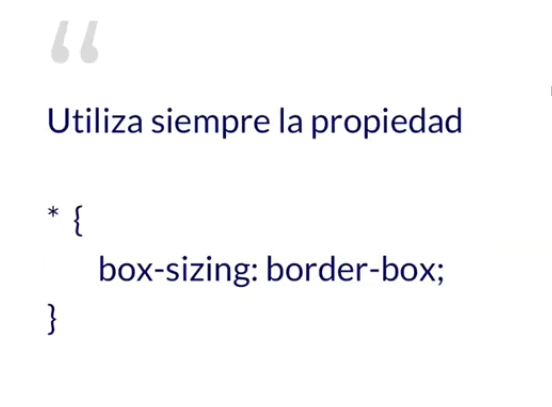
background-color: gray;

***/\*Valores.***

Como su nombre lo dice es el valor de la declaración (gray; 10px; etc)

background-color: gray;

***Para empezar…***



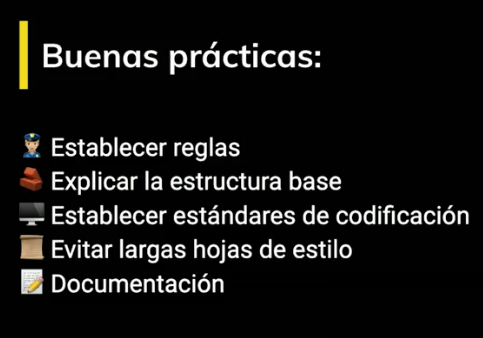
# ¿Qué son y para qué nos sirven las arquitecturas CSS?

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.

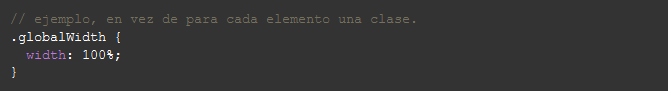
Buenas practicas

* 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 código.



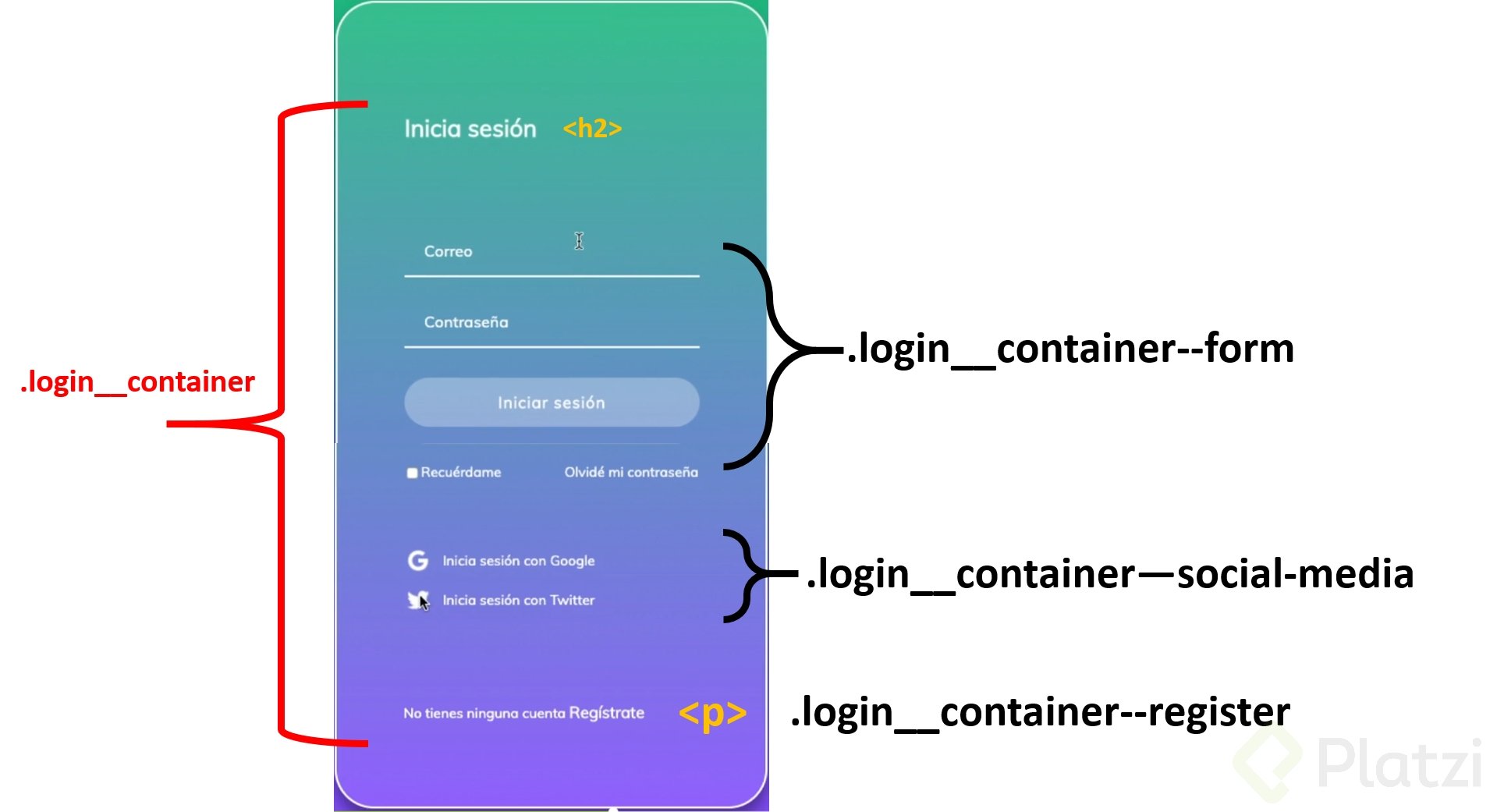
# OOCSS, BEM, SMACSS, ITCSS y Atomic Design.

**OOCSS**  
css orientado a objetos. Separa el diseño del contenido, así podemos reutilizar nuestro código



**BEM**  
Block Element Modifier. Separa los elementos y los modificadores

-**Bloque**: login  
-**Elemento**: login\_\_container  
-**Modificador**: Login\_\_container–form, –social-media y --register



**SMACSS**  
Scalable and Modular Arquitecture for CSS. Arquitectura de css escalable y modular. Para eso se realizan cinco pasos

1. Dividir nuestro css en componentes base
2. Definir Layout. Elementos que se utilizan solo una vez. ej. Footer, Header
3. Definir Módulos. Componentes que se usan más de una vez
4. Definir Estados. Estos estados definen los cambios que existen en nuestros elementos/componentes. **ej**. Cambio de color cuando hacen hover
5. Definir Temas. Separar los temas y sus cambios. Si tuvieras temas

**ITCSS**  
Inverted Triangule CSS. Triangulo Invertido de CSS. Lo que nos indica esta metodología es poder dividir todos nuestros archivos de css en ciertas partes para que no se mezclen.  
Triangulo invertido, desde arriba hacia abajo:

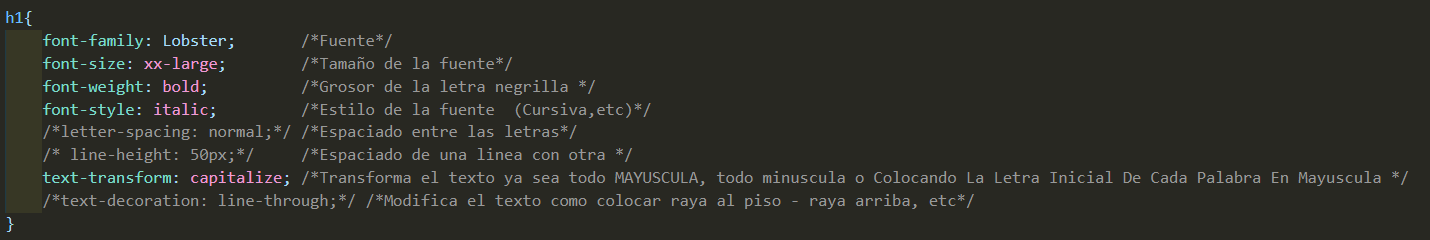
* Ajustes
* Herramientas
* Genérico
* Elementos
* Objetos
* Componentes
* Utilidades

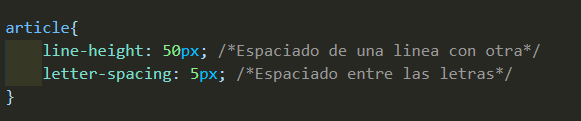
**Atomic Design**  
Basados en la química.  
Átomos < Moléculas < Organismos < Templates < Paginas

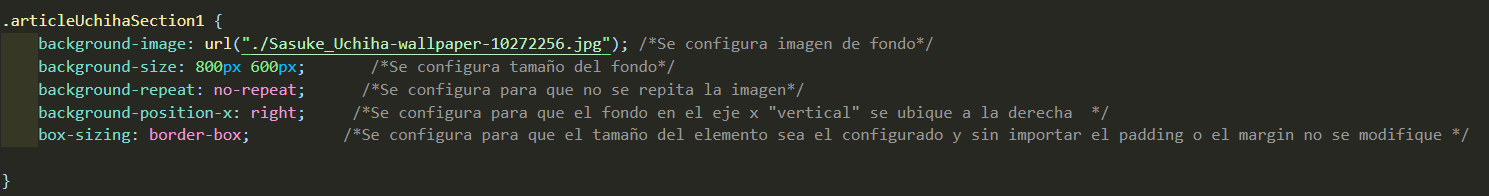
La elección de la metodología depende del proyecto y del equipo

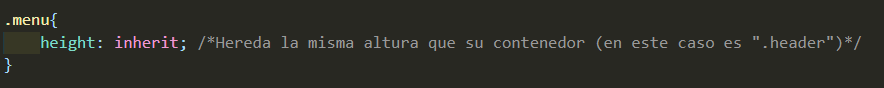
**Links**  
<https://www.smashingmagazine.com/2011/12/an-introduction-to-object-oriented-css-oocss/>  
<http://getbem.com/introduction/>  
<http://smacss.com/>  
<https://www.xfive.co/blog/itcss-scalable-maintainable-css-architecture/>  
<https://bradfrost.com/blog/post/atomic-web-design/>

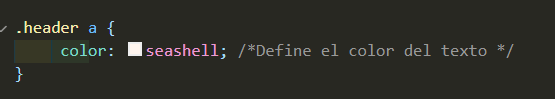
***Algunas propiedades CSS.***





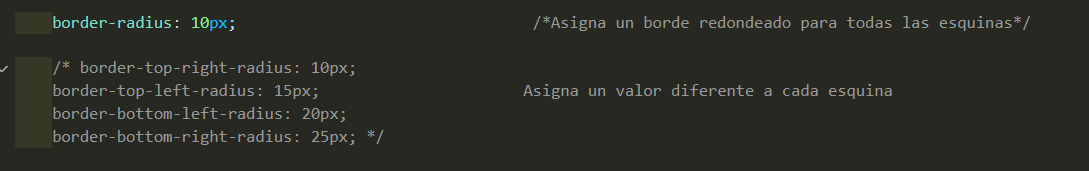


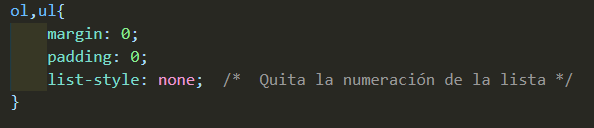


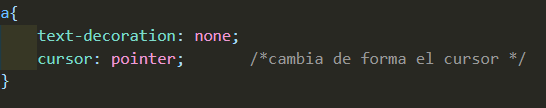


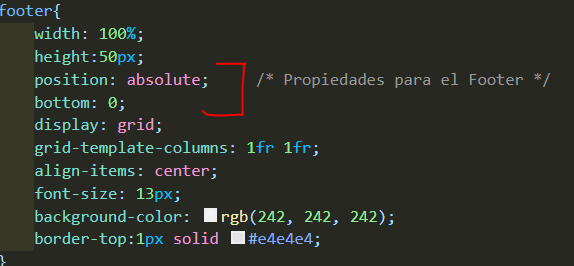


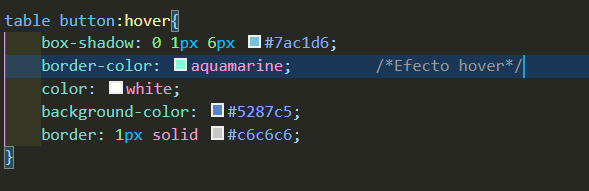


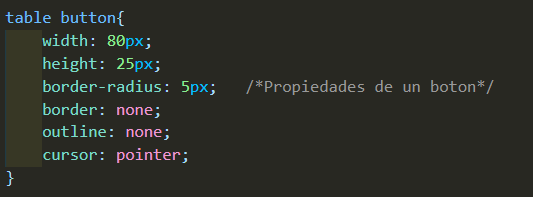










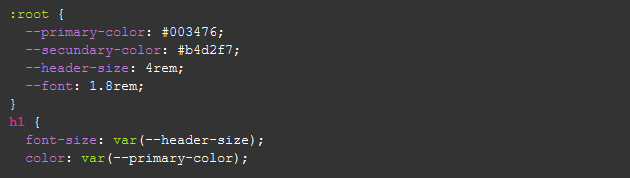


# Variables

Guarda valores que siempre usamos en los estilos para no repetir el mismo código a cada rato.

Se guardan en :root {}. Se escriben así: --nombreVariable: valor;

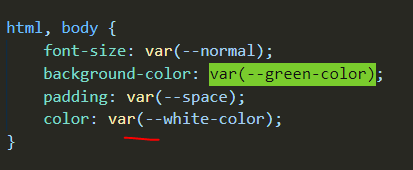
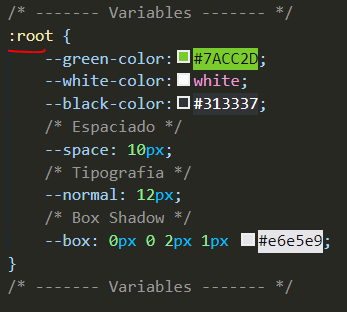
Para llamarlas se pone en el valor de la propiedad: var(--nombreVariable);



# Variables de CSS

Las Variables de CSS nos permiten centralizar los valores repetitivos por todo nuestro CSS, vamos a utilizar estas funcionalidades para ahorrarnos mucho si queremos cambiar el valor de algún color o tamaño de nuestra página.

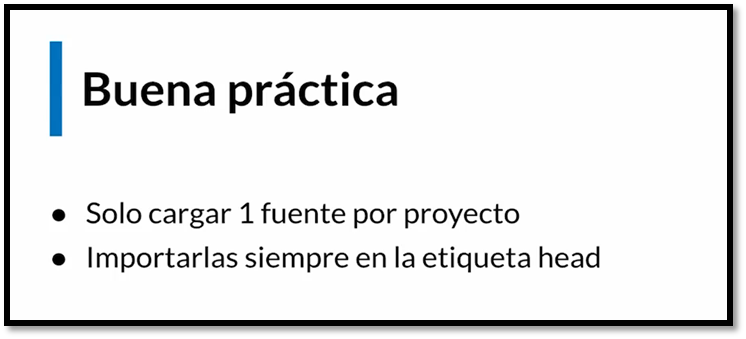
***Declaración. Llamado.***



# Fuentes personalizadas. Web fonts.

Vamos a utilizar fuentes personalizadas, para esto debemos ir a [Google Fonts](https://fonts.google.com) y elegir la que mejor se acomode a nuestro diseño.





Según conceptos de diseño hay varias recomendaciones para elegir la fuente / tipografía:

* Para textos en digital (Como páginas web) usamos fuente sans-serif sin esas terminaciones (serifas).
* Las fuentes con Serifa se recomiendan para impresas o en digital para títulos.

Para calcular el interlineado usamos la siguiente regla:

“2 puntos más que la fuente (Fuente de 8px pues interlineado de 10px)”

# 

# Pseudo clases y pseudo elementos.

**Pseudo clases**

Define el estilo de un estado especial de un elemento.

Agregar al final de la clase :nombreAccion

# 

**Pseudo Elementos**

Define el estilo de una parte específica de un elemento.

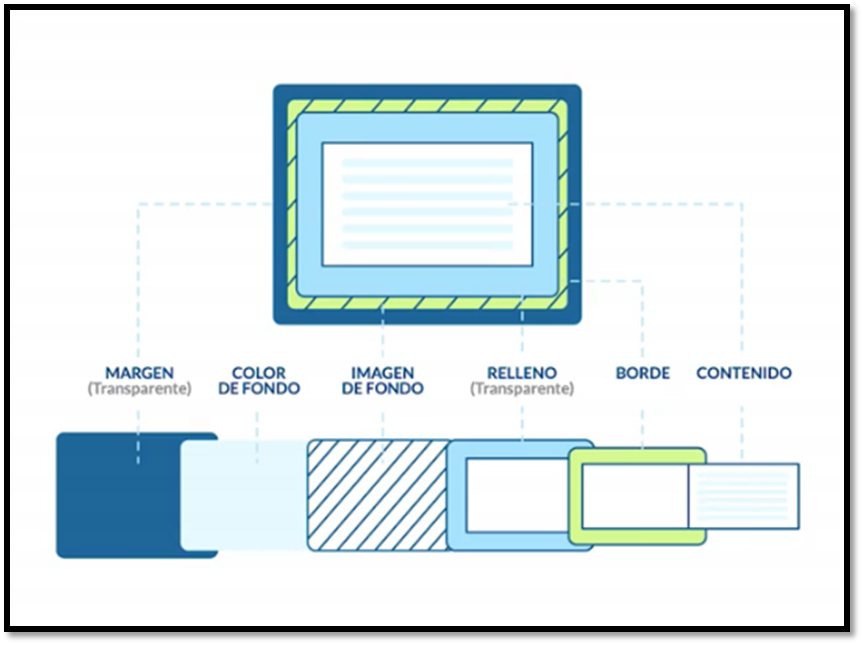
Agregar al final de la clase ::nombreAccion

# 



# Modelo de caja





# Box-sizing.

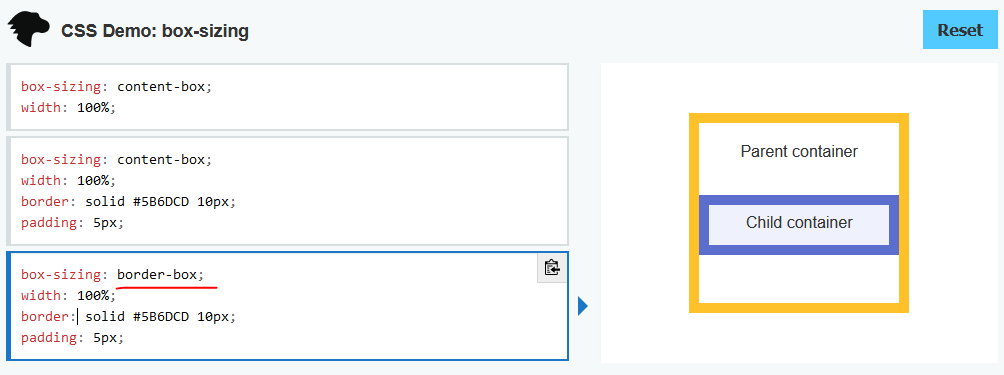
# Box-sizing: border-box hace que se calcule automáticamente el tamaño del elemento con el *padding* y el *border*, para que de esta manera no se salga de mi navegador generando scroll.

Por defecto en el [modelo de caja de CSS](https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Box_Model/Introduction_to_the_CSS_box_model), el ancho y alto asignado a un elemento es aplicado solo al contenido de la caja del elemento. Si el elemento tiene algún borde (border) o relleno (padding), este es entonces añadido al ancho y alto a alcanzar el tamaño de la caja que es desplegada en pantalla. Esto significa que cuando se definen el ancho y alto, se tiene que ajustar el valor definido para permitir cualquier borde o relleno que se pueda añadir.

La propiedad box-sizing puede ser usada para ajustar el siguiente comportamiento:

* **content-box** es el comportamiento CSS por defecto para el tamaño de la caja (box-sizing). Si se define el ancho de un elemento en 100 pixeles, la caja del contenido del elemento tendrá 100 pixeles de ancho, y el ancho de cualquier borde o relleno se le añadirá al ancho final desplegado.
* **border-box** le dice al navegador tomar en cuenta para cualquier valor que se especifique de borde o de relleno para el ancho o alto de un elemento. Es decir, si se define un elemento con un ancho de 100 pixeles. Esos 100 pixeles incluirán cualquier borde o relleno que se añadan, y la caja de contenido se encogerá para absorber ese ancho extra. Esto típicamente hace mucho más fácil dimensionar elementos.

# 

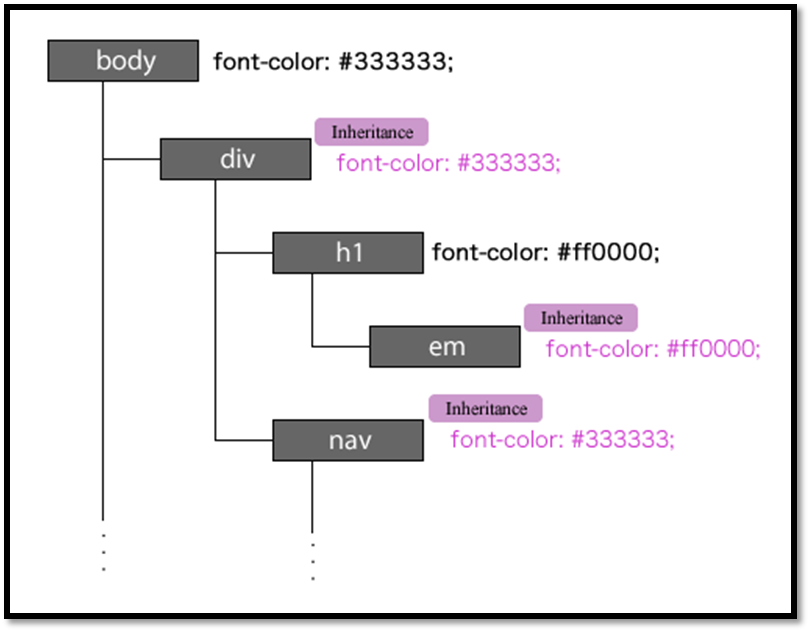


# Herencia

**Inherit**. Este es un valor por medio de una keyword que especifica que, a la propiedad que se la apliquemos debe de heredar los valores de su elemento padre. Podemos decir que la palabra **Inherit** significa “Usa el valor de mi padre”, si el elemento padre no tiene definido dicho valor el navegador seguirá el DOM hasta que encuentre un elemento superior que lo contenga, y en última instancia de no tenerlo ningún elemento superior se aplicara el valor por defecto.

**Initial**. Este valor pertenece a la especificación CSS3 y cuando aplicamos a una propiedad el valor initial estamos dando el valor inicial y predefinido por el navegador en cuestión.

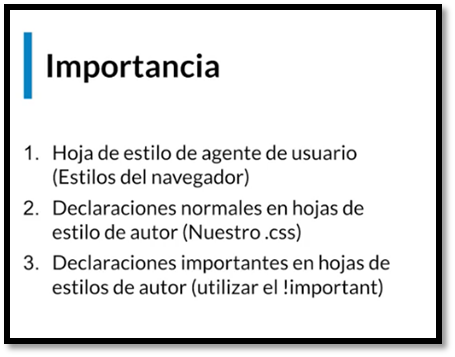
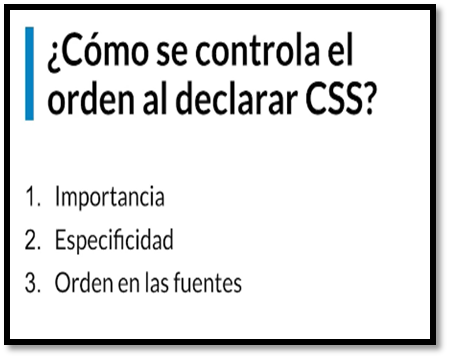
**Unset**. Este valor unset es una combinación entre inherit y initial, cuando utilizamos este valor en una propiedad esta tratará de heredar el valor de su elemento padre si este está disponible, de no ser así este valor colocará el valor de la propiedad en su valor inicial, como si usáramos inherit e initial juntos.



# Especificidad en selectores.

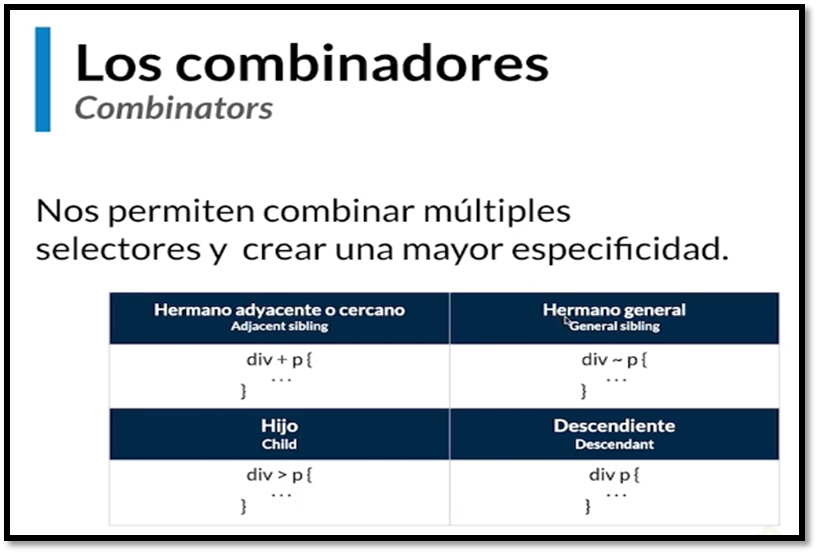
# Es importante conocer como es el orden en que los estilos van a ser cargados primero para determinar esos bugs en los que no sabemos porque se aplican estilos no esperados.

# El navegador los primeros estilos que va a cargar son los que tiene por defecto el navegador, luego cargara por orden nuestros archivos css linkeados y por últimos nuestros estilos en línea y los ¡important.



# Combinadores:



# Hermano Adyacente (combinator)

# 

# Hermano General (combinator)

# alt + 123

# Hijo (combinator)

# 

# Descendiente (combinator)

# 

**Juego para practicar selectores:**

<https://flukeout.github.io/>

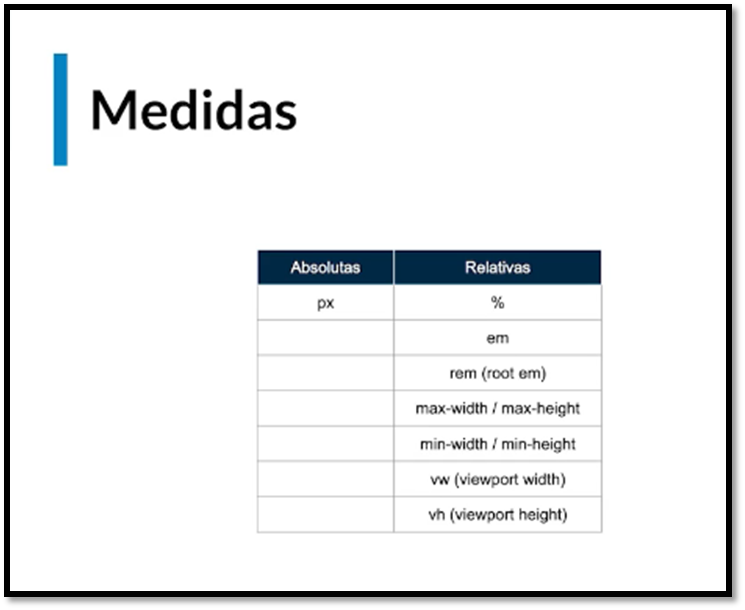
**Sheet Cheat Emmet.**



# Medidas.

# Medidas Absolutas: el valor de este no cambia y siempre será el mismo, así la página cambie su tamaño.

# Medidas Relativas: estas medidas heredan el tamaño o se basan en el tamaño de su elemento padre.



# Medidas REM y su uso con Font-size

La medida relativa REM funciona igual que el EM, con la diferencia que es relativo al valor de la fuente del elemento **html**, y no tiene en cuenta el valor heredado o del elemento que lo contiene.

Por **defecto** el html viene con un tamaño de fuente de **16px**.  
.  
1 REM = 16PX  
.  
Si queremos aplicar rem de una forma más sencilla para no tener que hacer tantos cálculos hacemos lo siguiente:

  
.  
Esto lo que hará es darle un valor de 10px ya que 16px - 62.5% = 10px

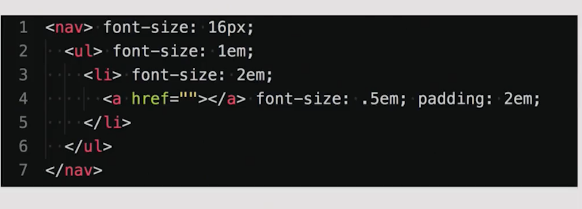
Ahora si por ejemplo a una etiqueta le asignamos **2rem** este hará referencia a **20px**, o si por ejemplo le damos un valor de **1.5rem** su valor será de **15px**

**Como buenas prácticas.**

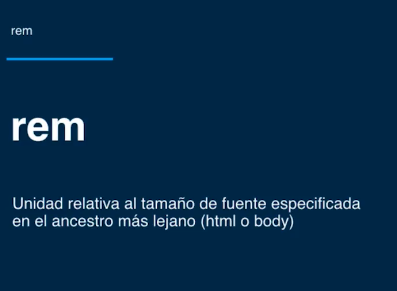


# Medidas relativas útiles en Responsive Design.

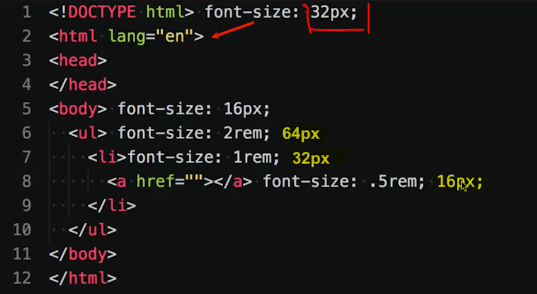
Lo primero que debes tener en cuenta es que estas medidas son maleables, en la medida en que dependen de su fuente de origen o medida madre. Entre ellas se encuentran el porcentaje (longitud referente al tamaño de los elementos padre), los em (unidad relativa al tamaño de fuente especificada más cercano), los rem (unidad relativa al tamaño de fuente especificada en el ancestro más lejano, como html o body) y tamaños del viewport **vw**/**vh** (longitud relativa porcentual con respecto al viewport).

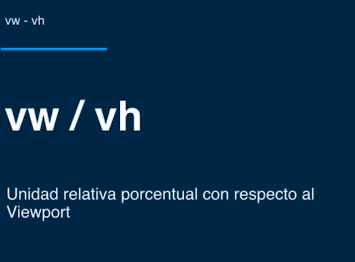
 





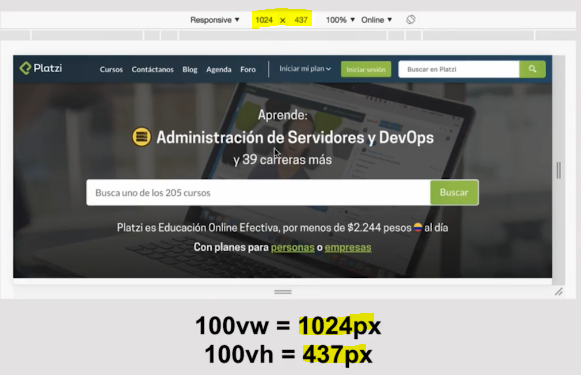






vw=”**V**iewport-**W**idth”

vh=”**V**iewport-**H**eight”



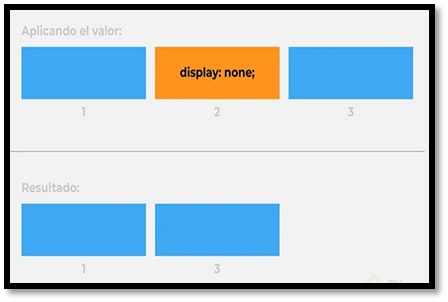
# Displays

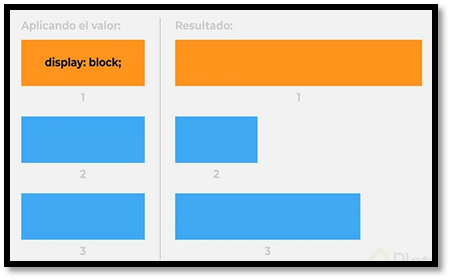
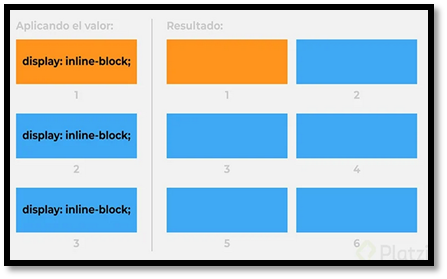
**Block**: Estos toman el 100% del width, por lo que un elemento no puede posicionarse a un lado de él.  
Se le puede poner el width deseado, height deseado, añadir margin, padding sin problema. Pero recordando que ocupara este elemento todo el largo de una Fila por así decirlo.

**Inline**: Estos elementos solo ocuparan el ancho dependiendo de su contenido. Por lo tanto, estos elementos si permiten que, si un elemento cabe a lado suyo, se posicione este ahí sin problemas.  
Las **desventajas** es que no se les puede modificar el **width**, **height**, ni colocar **margin** u **padding** tanto top, como bottom.

**Inline-block**: Este tiene la combinación de los 2 anteriores. Haciéndolo un mejor candidato para usarlo.  
Permite modificar su width, height, añadirle margin, padding sin problemas y lo mejor es que mientras que haya espacio a un lado suyo, este permitirá posicionar más elementos si encuentra espacio.

**None: Este display nos permitirá ocultar elementos.**

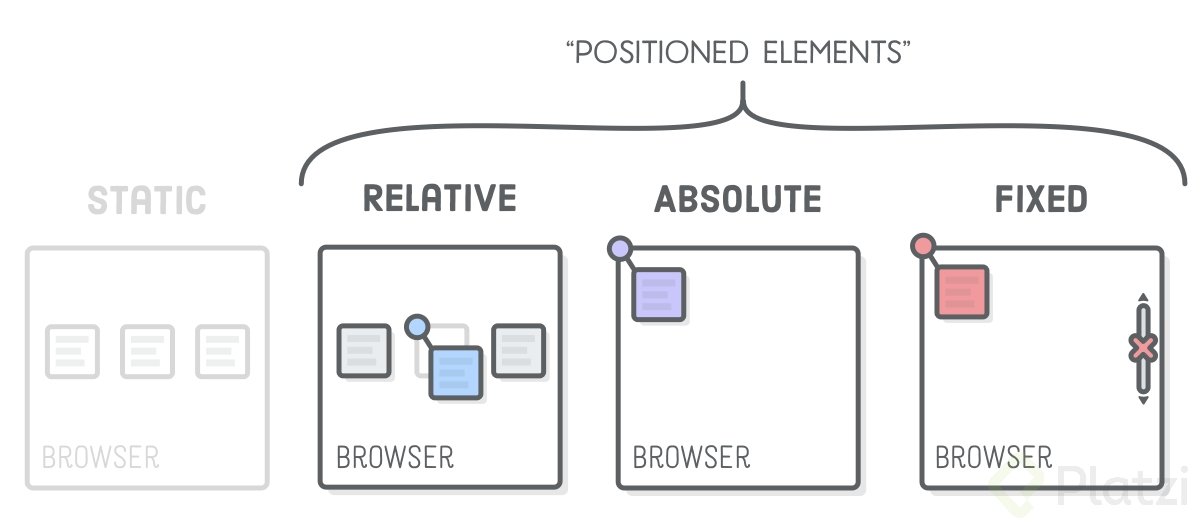
 

## **CSS Positions**

Es importante tener presente:

* Todos los elementos mantienen un flujo dentro del HTML.
* Este flujo casi siempre se rompe debido al posicionamiento que queremos darles a los elementos.
* Todos los elementos por defecto son ***Static. “No posicionado”***
* Al manejar elementos posicionados se desbloqueará el ***z-index*** que nos servirá para la superposición de elementos.
* El ancho de un elemento posicionado está definido por su contenido.



* **Static** (la posición por defecto): **NO** se considera un elemento posicionado porque es la posición por defecto dentro del HTML, este es el posicionamiento predeterminado y cuando encuentren documentación static se considera como un elemento **NO** posicionado.

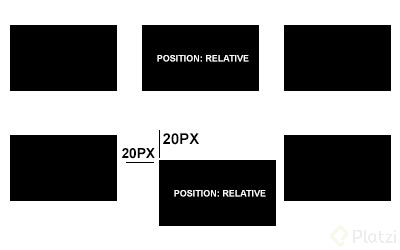
Con las otras opciones, se activan las propiedades de **top, bottom, left, right** y **z-index**.

* **Relative**: el objeto se mueve en base al lugar donde se encuentra originalmente. Cuando se coloca un elemento en relative no ocurre un cambio a primera vista sin embargo cuando se mueve el elemento, el DOM que es todo el documento HTML RESPETA el espacio que está ocupando dicho elemento.

El elemento es posicionado de acuerdo con el flujo normal del documento, y luego es desplazado con relación a sí mismo, con base en los valores de top, right, bottom, and left. El desplazamiento no afecta la posición de ningún otro elemento; por lo que, el espacio que se le da al elemento en el esquema de la página es el mismo como si la posición fuera static.

*Su posición va ser* ***relativa*** *a su posicionamiento en el flujo del HTML*

*Mantiene el vínculo.*

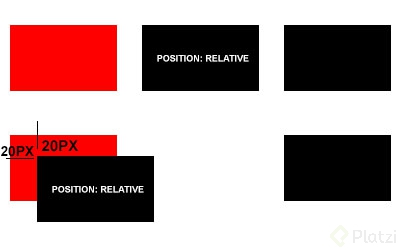


* **Absolute**: el objeto se ubica de manera absoluta con el elemento más cercano que tenga posición relativa o con el body. Los valores toman referencia del ancestro POSICIONADO más cercano quiero decir del **ancestro** que tenga alguna de estas propiedades establecidas.

El elemento se ubica relativo al bloque contenedor inicial. Su posición final está determinada por los valores de [top, right, bottom, y left].

Este valor crea un nuevo contexto de apilamiento cuando el valor de z-index no es auto. Elementos absolutamente posicionados pueden tener margen, y no colapsan con ningún otro margen.

***No*** *mantiene el vínculo con el espacio del elemento en el flujo del HTML.*

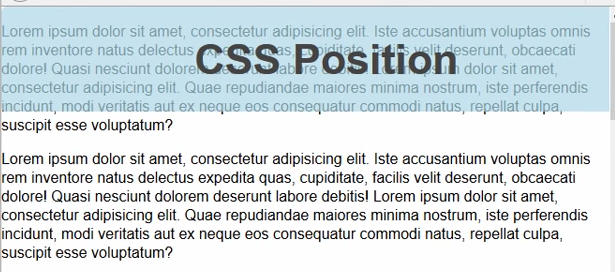


* **Fixed**: El elemento se muestra de manera fija en el viewport. El elemento con un valor fijo permanecerá fijo en la parte especificada de la página, incluso si este elemento se tira hacia arriba o hacia abajo. Podemos determinar la ubicación de este elemento con las propiedades de [top, right, bottom, y left].

Por ejemplo, si queremos que el elemento se ubique en la parte inferior derecha de la página, será suficiente escribir los siguientes códigos CSS.

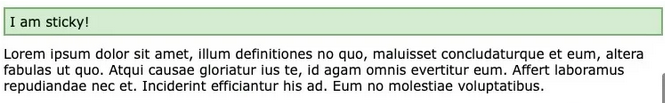


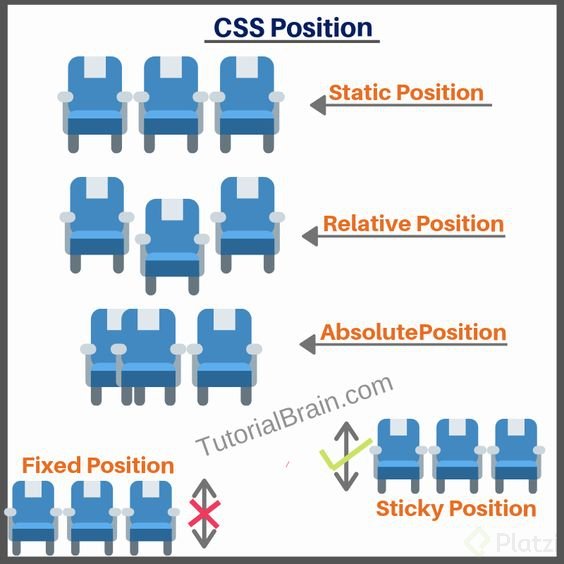
*El elemento pierde su espacio en el flujo por lo que pueden quedar elementos superpuestos.*



* **Sticky**: El elemento se queda de manera fija una vez que aparece en pantalla. Un elemento con un valor fijo regresa entre propiedades relativas y fijas. Mientras está de pie de acuerdo con la ubicación de la página, si intenta deshacerse de este elemento mientras desplaza su página hacia arriba o hacia abajo, comienza a actuar como un elemento fijo y cuando el elemento llega a su posición original, continúa actuando como un elemento relativo nuevamente.

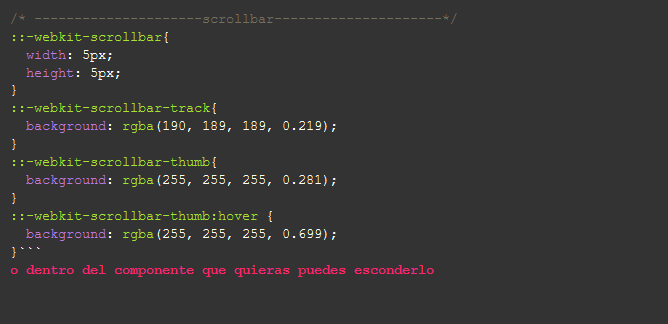
Para dar un ejemplo de esto:





# Estilos a Scroll.

# 



**Responsive Design.**

**Responsive Design** consiste en crear estilos que se adapten a cualquier tamaño y posición de nuestros dispositivos electrónicos. Para esto, la mayoría de los elementos organizados horizontalmente deben pasar a organizarse verticalmente.

La forma de añadir código CSS que se ejecute para tamaños de pantalla específicos es la siguiente:

@media (max-width: 600px) { /\* 600px es solo un ejemplo \*/

/\* Todos nuestros estilos responsive \*/

}

Normalize CSS es una libreria que nos resetea algunos valores por defecto del navegador para no tener problemas a futuro, muchos frameworks están basados en normalize CSS.

[**https://necolas.github.io/normalize.css/**](https://necolas.github.io/normalize.css/)

**Conceptos elementales de Responsive Design**

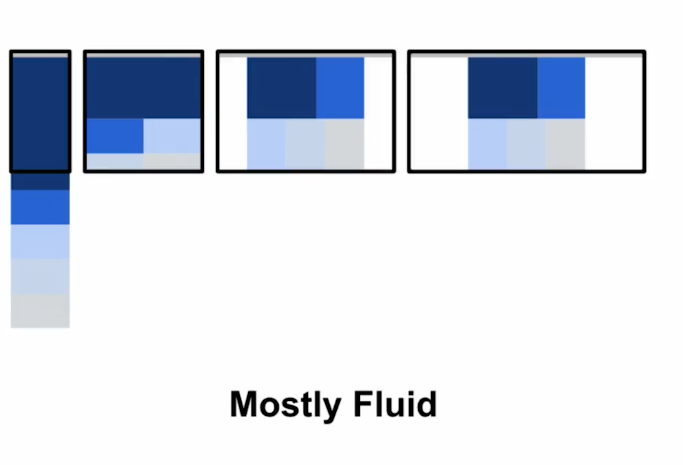
Para abordar el campo del Responsive Design es necesario que tengas claridad sobre algunos conceptos básicos.

Por este motivo, conoceremos qué medidas son necesarias aplicar para lograr que tus proyectos se adapten a pantallas de diversas medidas y condiciones, cuáles son los principios del Responsive Design (mostly fluid, colocación de columnas, layout shifter, tiny tweaks, off canvas).

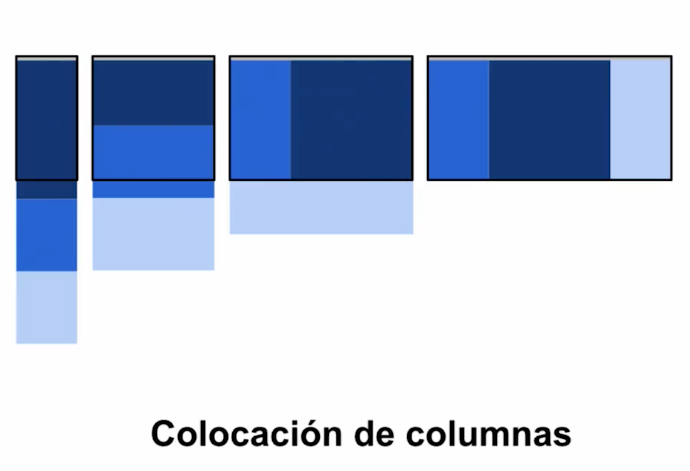
Finalmente, aprenderás el objetivo principal del Responsive Design: la óptima visualización de las web sites en cualquier dispositivo y podrás tener referentes en [www.mediaqueri.es](http://www.mediaqueri.es) .

**Patrones en responsive design.**

**Mostly Fluid:**  
El patrón Mostly fluid consiste, principalmente, en una cuadrícula fluida. Por lo general, en las pantallas grandes o medianas se mantiene el mismo tamaño y simplemente se ajustan los márgenes en las más anchas.  
En las pantallas más pequeñas, la cuadrícula fluida genera el reprocesamiento del contenido principal, mientras que las columnas se apilan verticalmente. Una de las mayores ventajas de este patrón es que, en general, solo se necesita un punto de interrupción entre las pantallas grandes y las pequeñas.



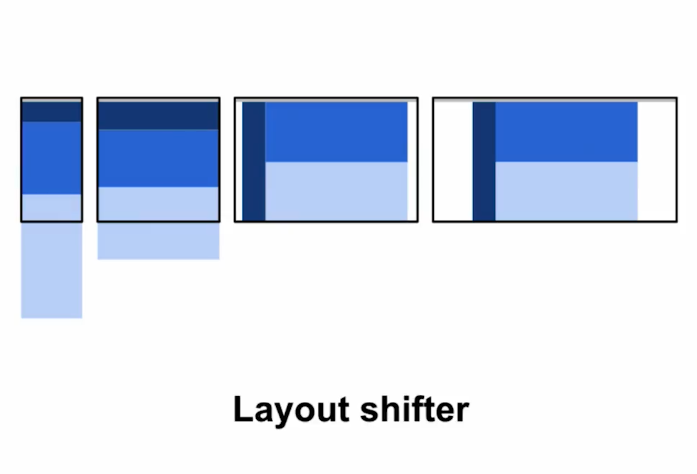
**Colocación de columnas:**  
En el caso de los diseños con varias columnas de ancho completo, durante el proceso de colocación de columnas éstas únicamente se colocan de forma vertical debido a que el ancho de la ventana es demasiado reducido para el contenido.  
En un momento dado, todas las columnas se apilan verticalmente. La selección de puntos de interrupción para este patrón de diseño depende del contenido y cambia para cada diseño.



**Layout shifter:**

El patrón Layout shifter es el más adaptable, ya que posee varios puntos de interrupción en diferentes anchos de pantalla.

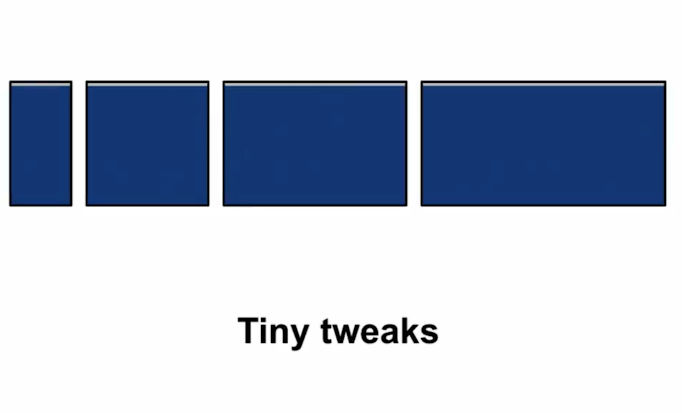
La clave para este diseño es el desplazamiento del contenido, en lugar de su reprocesamiento y colocación debajo de otras columnas. Debido a las diferencias significativas entre cada punto de interrupción principal, es más complejo de mantener, y es posible que se deban realizar cambios dentro de los elementos, no solo en el diseño de contenido general.



**Tiny tweaks**:

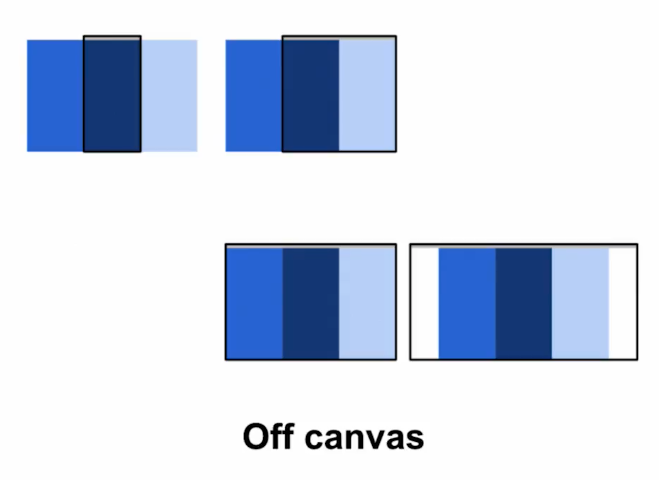
El patrón Tiny tweaks permite realizar pequeños cambios en el diseño, como ajustar el tamaño de la fuente, cambiar el tamaño de las imágenes o desplazar el contenido de maneras muy poco significativas.

Funciona correctamente en diseños con una sola columna, como los sitios web lineales de una sola página y los artículos con mucho texto.

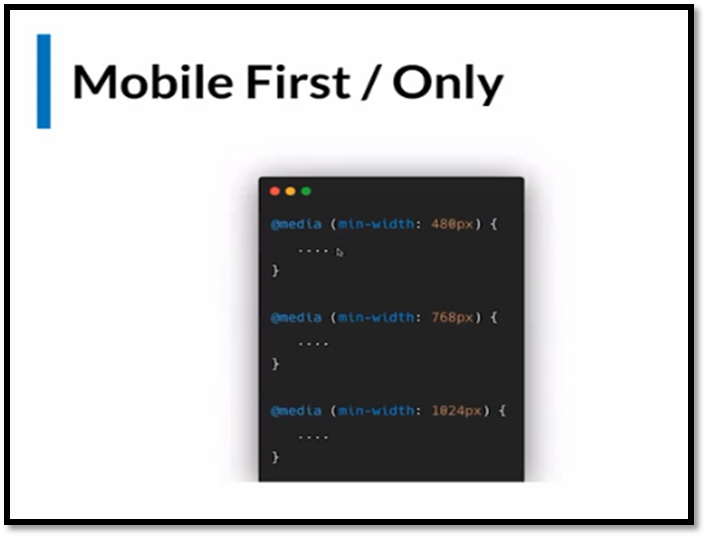


**Off canvas**:

En lugar de apilar contenido verticalmente, el patrón Off canvas coloca contenido menos usado (tal vez menús de navegación o de apps) fuera de la pantalla y solo lo muestra cuando el tamaño de la pantalla es suficientemente grande. En las pantallas más pequeñas, el acceso al contenido es posible con solo a un clic.



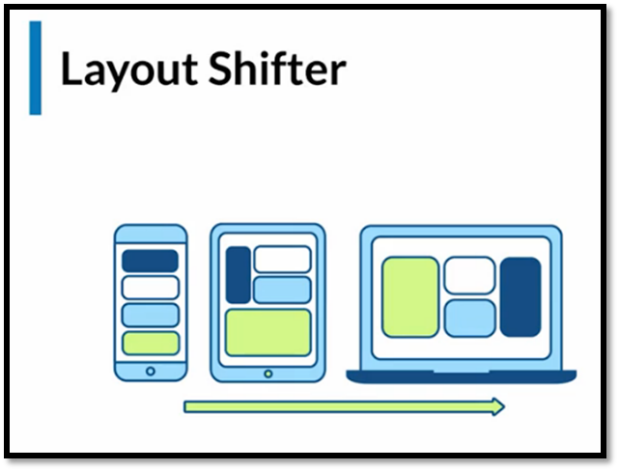
# Responsive design: media queries

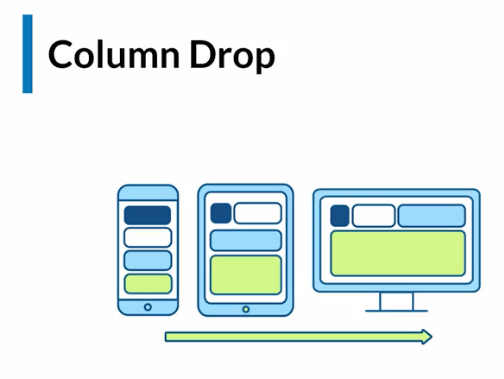




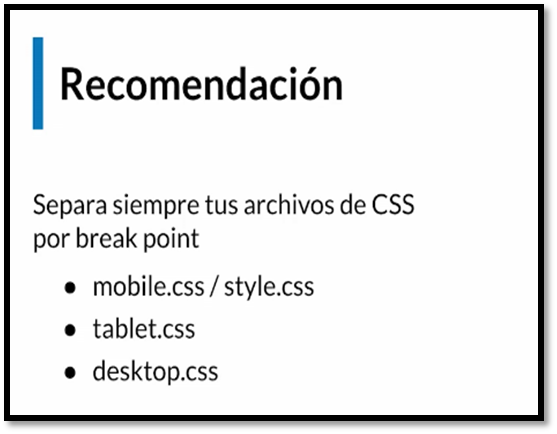
# Estrategias de responsive

# 

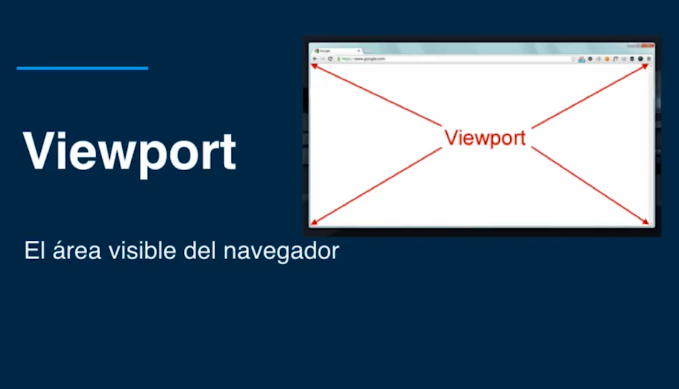
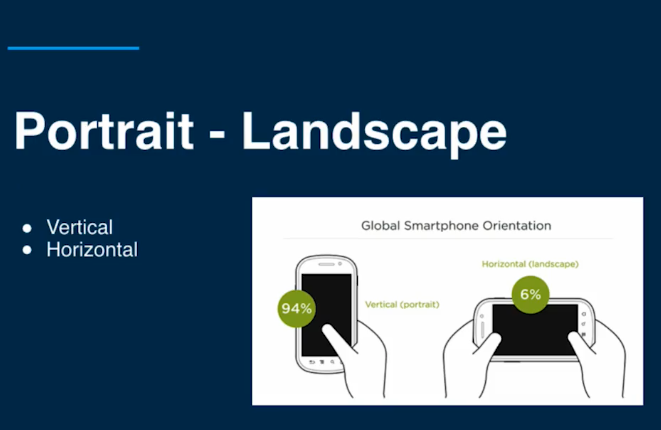




# Buenas prácticas y ejemplos de responsive



**Conceptos Importantes.**

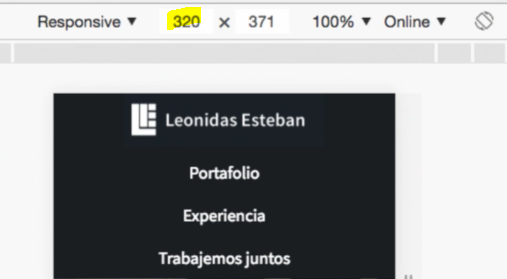
 

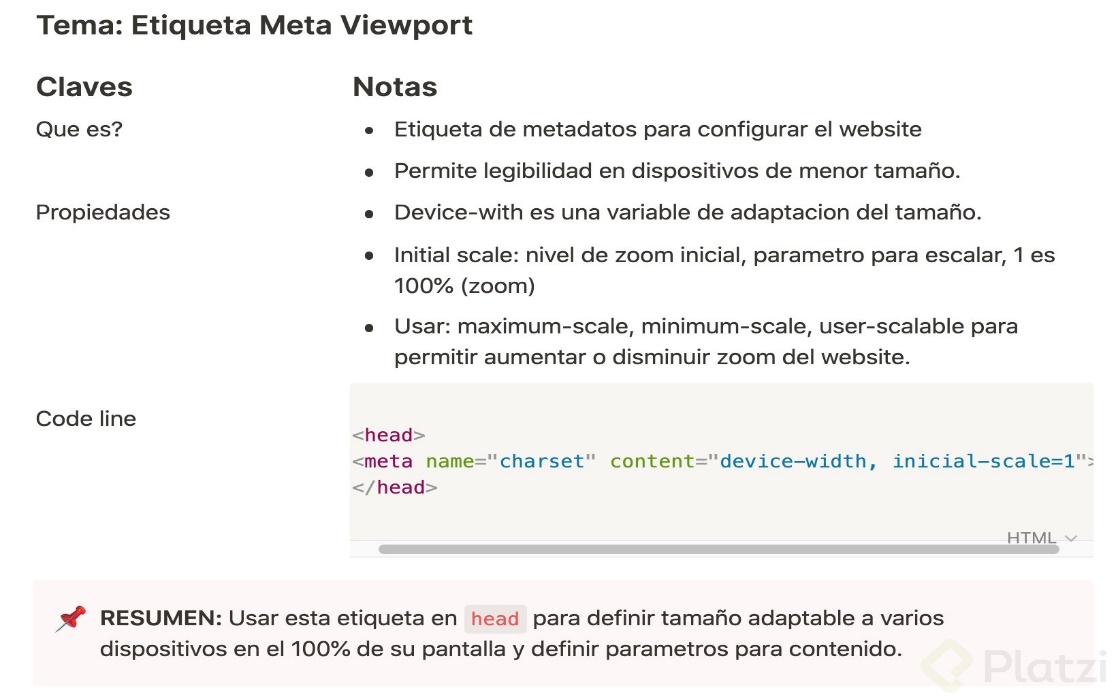
# Etiqueta meta viewport.

# Esta es una etiqueta de metadatos que te ayudará a configurar tu website para que sea visible en dispositivos de menor tamaño. Uno de los objetivos principales al usar esta etiqueta será que conserves la legibilidad de tu página web, al variar el escalado de tus contenidos.

La resolución estándar para una página Web Inicia desde 320px. En dispositivos actuales es difícil que se logre una resolución menor que eso.



width=device-width para que se adapte según la pantalla del dispositivo  
initial-scale=1.0 para indicar el escalado según el dispositivo



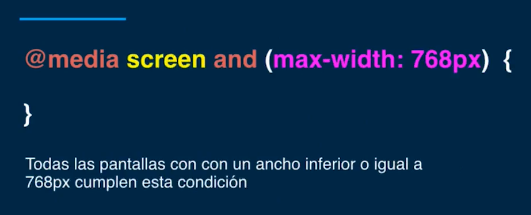
<meta name="viewport" content="width=device-width, initial-scale=1.0"/>

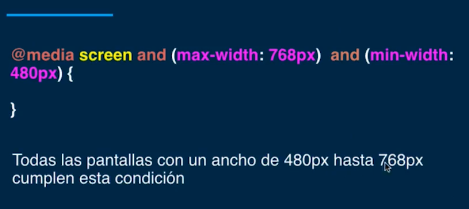
# Media queries

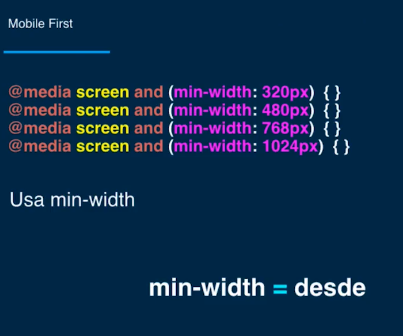
Para que logres los resultados que deseas en tus proyectos, es necesario cambiar ciertas propiedades para modificar el tamaño de los textos, contenidos y hojas de estilo; la manera de hacer esto es el media queries.

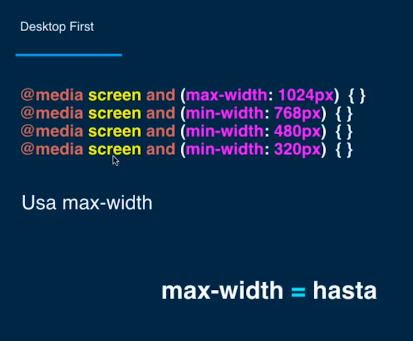
El media queries es un módulo de css que hace posible al responsive design, éste existe desde el 2010 y se encarga de adaptar la representación del contenido a características del dispositivo.











# Formas de incluir media queries

**PRIMERA FORMA:**  
Hacer una nueva hoja de estilos condicionando si se carga o no dicha hoja dependiendo de la condición del navegador.  
**Añadir esta línea de código en el <head> del archivo HTML**

<head>

      <link rel="stylesheet" href="css/media.css" media="screen and (max- width:768px)"/>

</head>

**SEGUNDA FORMA:**  
Agregar la expresión del Media Querie al final del código css:

**@media screen and (max-width: 768px) {**  
 [aquí se añaden los estilos css]  
**}**

**EJEMPLO:**

@media screen and (max-width: 768px) {

  body {

    border: 10px solid green;

    Background-color: red;

    }

  .ventana {

    border: 10px solid green;

    border-radius: 25px;

    }

  }

**TERCERA FORMA:**  
Abriendo unas etiquetas **<styles> </styles>** al final del head y dentro de ellas invocar los media queries.

<styles>

      @media screen and (max-width: 768px) {

        body {

          border: 10px solid green;

          Background-color: red;

          }

        .ventana {

          border: 10px solid green;

          border-radius: 25px;

          }

        }

      </styles>

**Medidas standard para el diseño responsivo en formato Desktop:**

@media screen and (max-width: 1024px) {

  [Estilos css para dispositivos cuyo ancho maximo son 1024px]

}

@media screen and (max-width: 768px) {

  [Estilos css para dispositivos cuyo ancho maximo son 768px]

}

@media screen and (max-width: 480px) {

  [Estilos css para dispositivos cuyo ancho maximo son 480px]

}

@media screen and (max-width: 320px) {

  [Estilos css para dispositivos cuyo ancho maximo son 320px]

}

# Ajustes responsivos

# Siempre es mejor tener las cosas en display: block y en width: auto porque da mucho mejor rendimiento en el render del navegador.

# Para hacer img responsives usar max-width: 100% Ajusta la imagen al tamaño del container. NO LO PONGAS EN EL MEDIA QUERY SI NO EN EL CODIGO CSS ORIGINAL

# Como buena práctica poner el width y el height en %, como un 50% o 100%, para tener buenos resultados a la hora de nuestro diseño responsivo

# Cuando hay unidades de márgenes o relleno, es recomendable hacer la sumatoria al momento de ajustar porcentajes a las unidades de ancho. Ejemplo: 48% width (para repartir entre 2 elementos), mas 1% de margin, que en total sumarian 48% + 48% + 1% + 1% +1% +1%.

# Con la tecla WINDOWS + [PUNTO] Abrimos los emojis en Windows 10.

# Para centrar objetos:

“***margin: 0 auto***” se utiliza normalmente para bloques como div, figure, o alguna otra etiqueta que sirve como contenedor

“***text-align: center***” nos permite centrar el contenido, como los encabezados, textos, imágenes dentro de párrafos, etc. que están dentro de sus respectivas cajas.

# Media queries con JavaScript

En esta clase aprenderás a implementar media queries con JavaScript, para esto usarás instrucciones como window.matchMedia, console.log -nuevamente-, event, entre otros.

El propósito es que tu menú quede listo para ofrecer una experiencia interactiva y sea flexible en distintos dispositivos, es decir, que sea interactivo y responsivo.

De forma adicional, aprenderás a agregar y quitar listeners de tus eventos, pues no siempre son la mejor opción en la experiencia de navegación.



# Animaciones y transiciones

Las animaciones nos permiten cambiar los estilos de nuestros elementos durante un tiempo. Para esto debemos configurar nuestra animación con algunos estilos iniciales y finales para aplicarla a alguno de nuestros selectores de CSS:

@keyframe fadeIn {

from {

opacity: 0;

}

to {

opacity: 1;

}

}

.fadeIn {

animation-name: fadeIn;

animation-duration: 4s;

animation-fill-mode: 4s;

}

No olvides repetir los atributos de animación o transiciones utilizando el prefijo -webkit-:

.fadeIn {

animation-name: fadeIn;

-webkit-animation-name: fadeIn;

animation-duration: 4s;

-webkit-animation-duration: 4s;

animation-fill-mode: 4s;

-webkit-animation-fill-mode: 4s;

}

La herramienta [Animate.css](https://daneden.github.io/animate.css/) puede ser muy útil para encontrar y probar diferentes animaciones/transiciones algo comunes en sitios 👌😉.

# Instalación de SASS y configuración inicial

Instalación de SASS con NPM:

npm install -g sass

Si usas Windows puedes usar el gestor de paquetes [Chocolatey Package Manager](https://chocolatey.org/) e instalar SASS con el siguiente comando:

choco install sass

Si usas Mac puedes usar [Homebrew](https://brew.sh) para instalar SASS con el siguiente comando:

brew install sass/sass/sass

[***https://github.com/teffcode/sass-workshop***](https://github.com/teffcode/sass-workshop)

# ¿Qué es un componente? Analicemos nuestros diseños

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.



# STORYBOOK: LA MEJOR HERRAMIENTA PARA USAR CON REACT

# <https://www.youtube.com/watch?v=lA-4Hz9N7qA>

# <https://www.youtube.com/watch?v=WgsaTWmzvFE>

**Accesibilidad.**

**Semántica.**

En nuestros proyectos será muy importante llevar la semántica de nuestros elementos HTML. Aquí podremos consultar la [Tabla Periódica para HTML](https://allthetags.com/) para que tener más contexto y mayor accesibilidad a la documentación en cuanto a semántica o todas las etiquetas existentes.

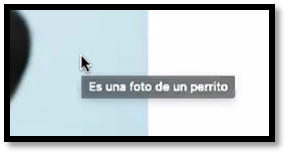
**Textos.**

Se recomienda fuertemente usar medidas relativas como rem, para poder incrementar el tamaño del texto para personas con visibilidad disminuida. Las opciones de navegador que cambian el tamaño de las fuentes no funcionan cuando las fuentes de html en el texto están en pixeles (px). Usar medidas relativas nos ayudara a que el texto escale de una manera mucho más adaptativa a todos los tamaños. **Importante uso de rem.**

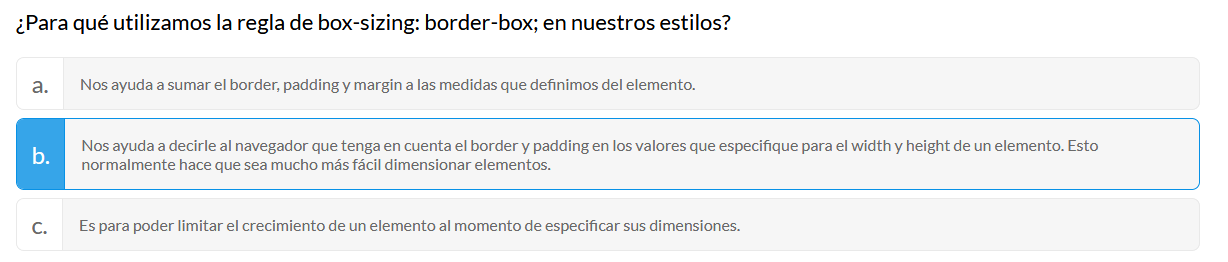
**Labels, alt y title.**

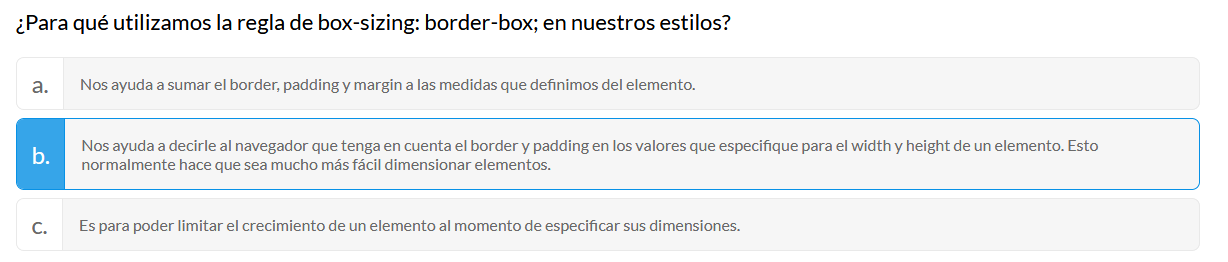
En la creación de formularios es importante el uso de la etiqueta **<label>** como contenedor de mi input y mi span ya que de esta manera el navegador ayudara a hacer un *autofocus* al input al momento en que de clic sobre mi label. No olvidar también siempre usar la propiedad **alt=””** en la etiqueta img como tema importante de accesibilidad.

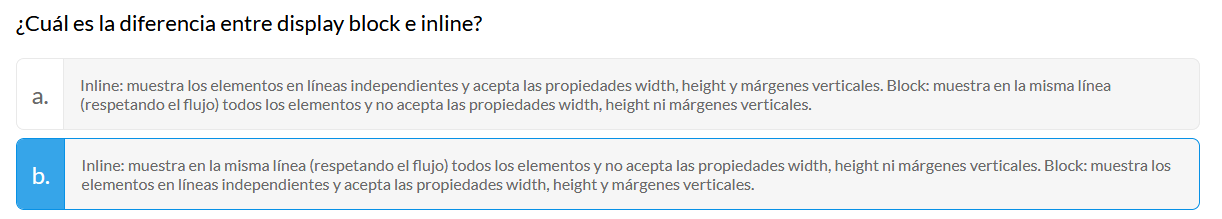
Por último, existe la propiedad **title=””** que se la puedo añadir a etiquetas **<a> <img>** que lo que me permite es que cuando ubique mi cursor sobre la imagen o ya sea un link me muestre ese *título* que he asignado.

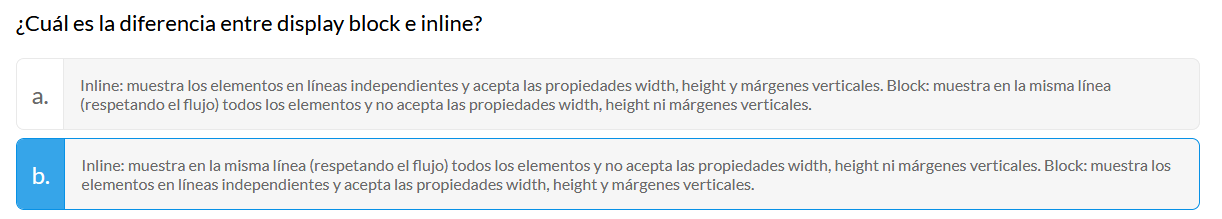


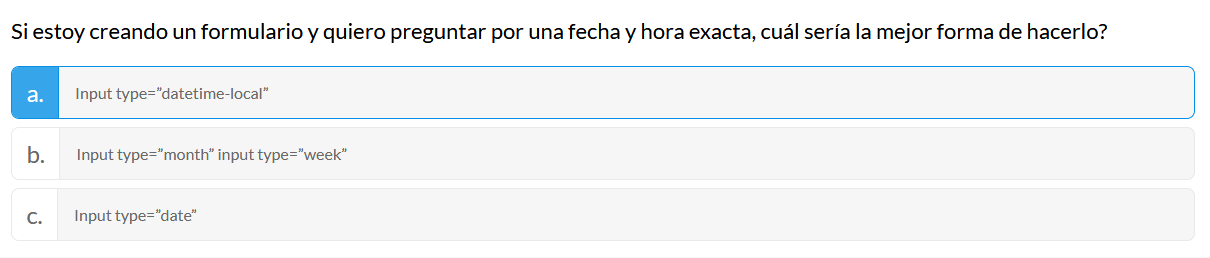
**Q &A IMPORTANTES.**



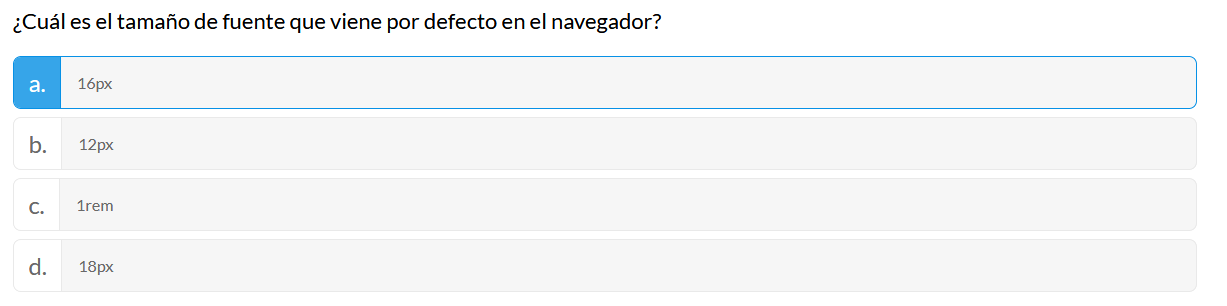


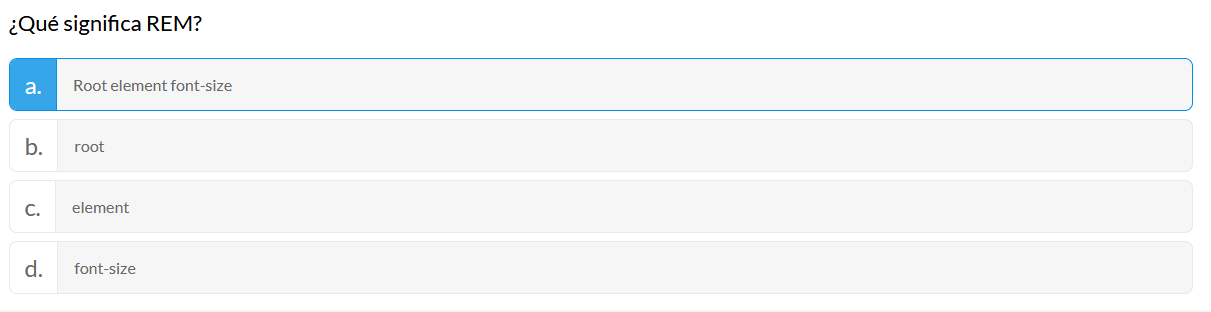


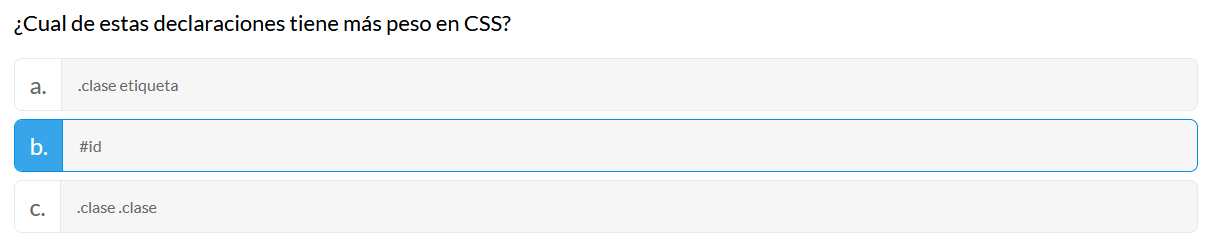


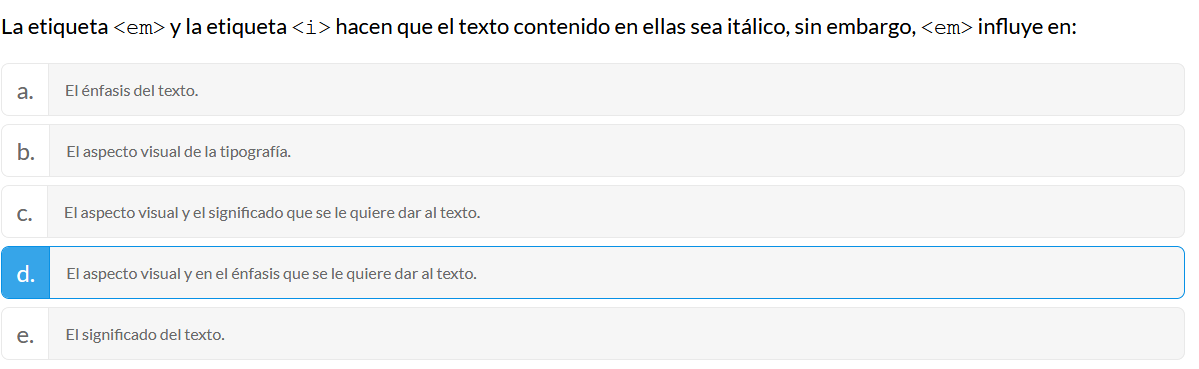


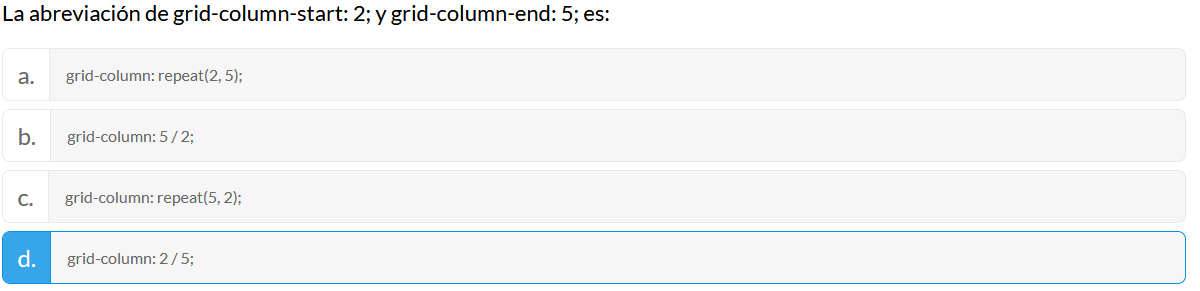


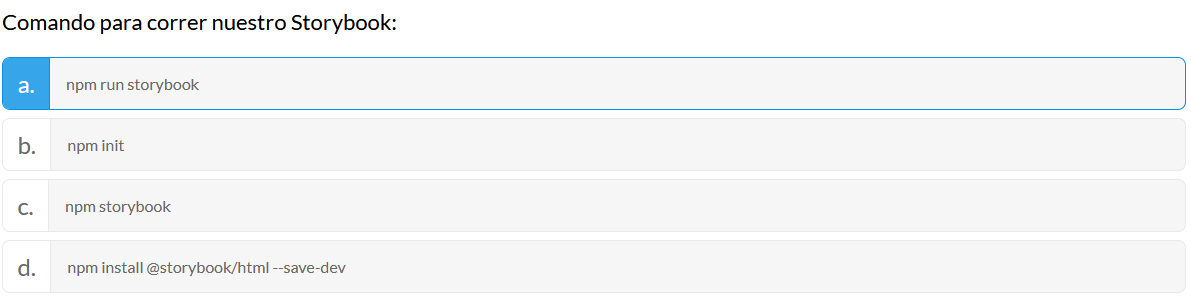












¿Con qué propiedad puedo darle tamaño a fuentes de íconos? Font-size

¿Cómo se le dice a un teléfono en modo vertical? Portrait

¿Cómo se le dice a un teléfono en modo horizontal? Landscape

De acuerdo con el siguiente código, ¿cuánto vale 1rem?

html { font-size: 16px; }

¿A qué reacciona este media query?

@media screen and (max-height: 768px) Pantallas hasta 768px de ancho

¿A qué reacciona este media query?

@media screen and (min-height: 768px) Pantallas desde 768px de alto

¿Con qué API de JavaScript puedo detectar media queries? matchMedia

¿A qué reacciona este media query?

@media screen and (min-width: 768px) Pantallas desde 768px de ancho

¿A qué reacciona este media query?

@media screen and (max-height: 768px) Pantallas hasta 768px de alto

¿Cuánto valdría 1.5em en el borde en .body ?

body { font-size: 16px }

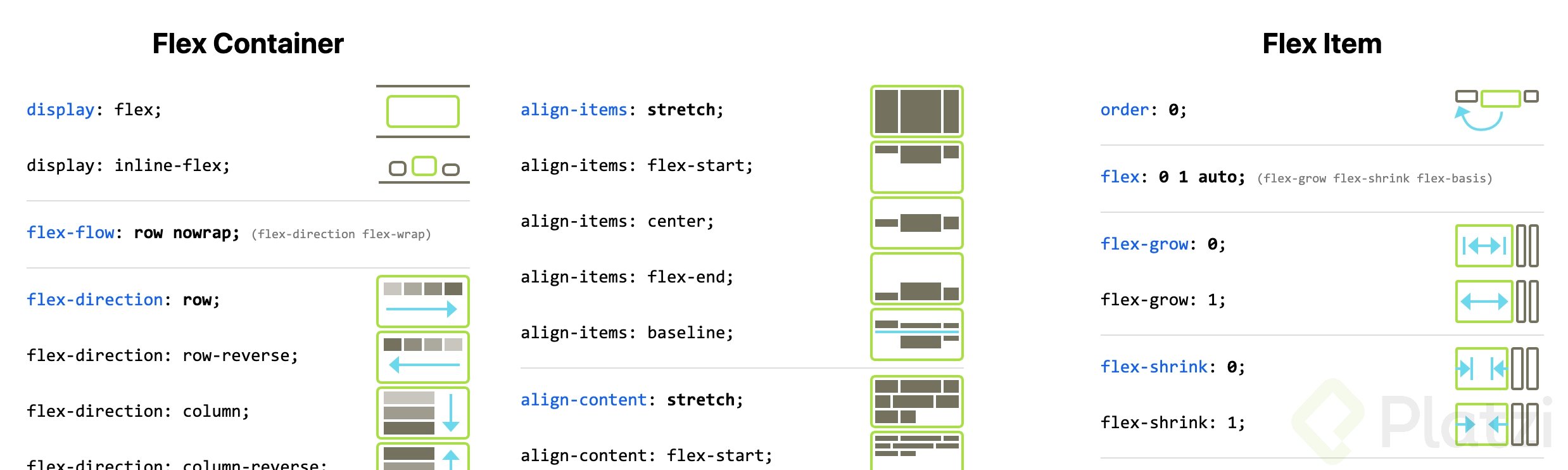
.body { font-size: 18px } 27px

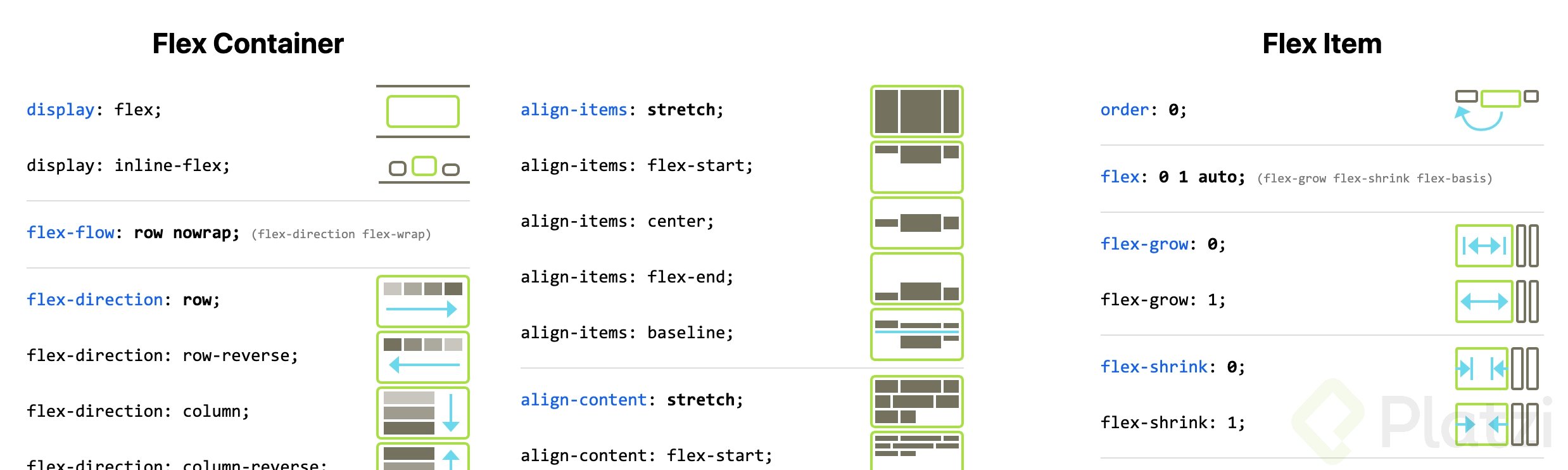
¿Cómo puedo restaurar al valor por defecto cualquier propiedad de CSS? Initial

# Flexbox.

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

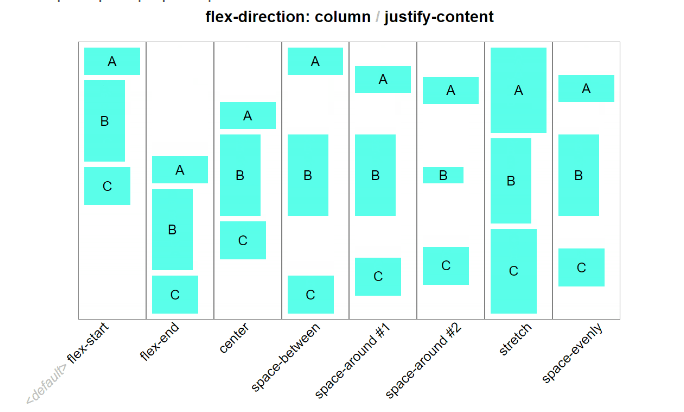
Es importante tener presente que tendremos un contenedor y los elementos que queremos organizar dependiendo de nuestras necesidades.





<https://darekkay.com/dev/flexbox-cheatsheet.html>

# Display Flex



***Paginas para practicar Grid y Flex.***

[***https://flexboxfroggy.com/#es***](https://flexboxfroggy.com/#es)

[***https://cssgridgarden.com/#es***](https://cssgridgarden.com/#es)

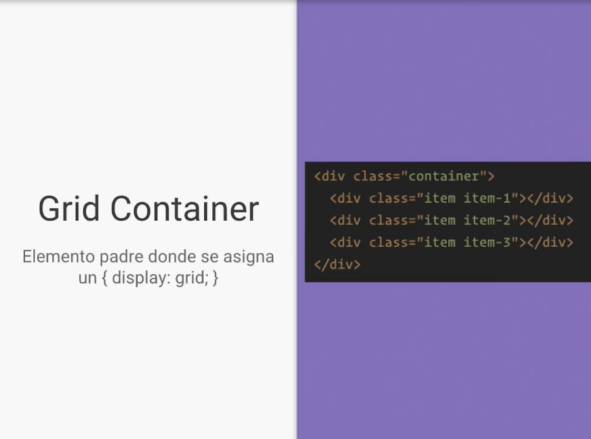
***CSS Grid Layout.***

**¿Qué es CSS grid layout?**

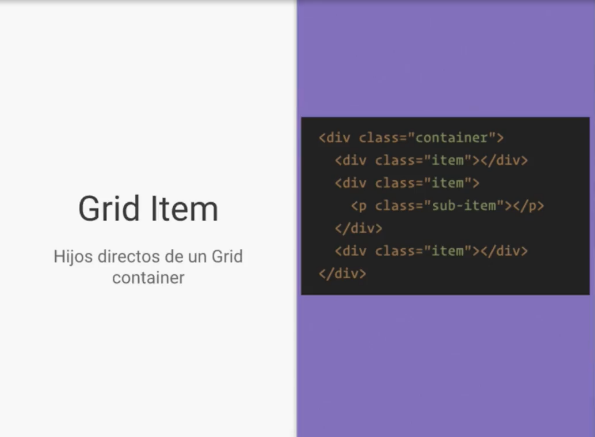
Es un sistema de rejilla para la maquetación web, no es un framework o una librería, es una adición al lenguaje que nos permite distribuir contenido de forma eficiente.

**Conceptos fundamentales sobre CSS Grid Layout.**

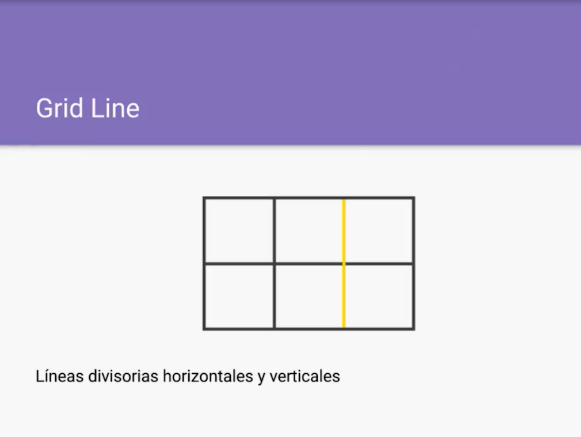
**Grid Container**: va a ser el elemento padre que va a tener puesto un nuevo tipo de display: **grid**. Nos permite colocar otras propiedades para manipular nuestro layout.



**Grid Item**: Son nuestros componentes, contenidos, los que vamos a manejar. Nuestras filas o columnas que vamos a mover a nuestro gusto. Son hijos directos de grid.



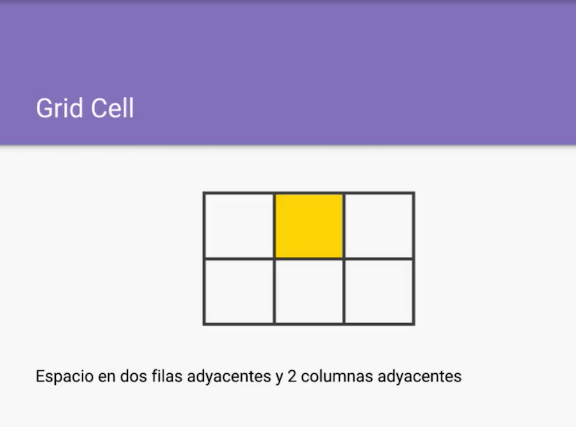
**Grid Line**: Líneas divisorias horizontales y verticales.



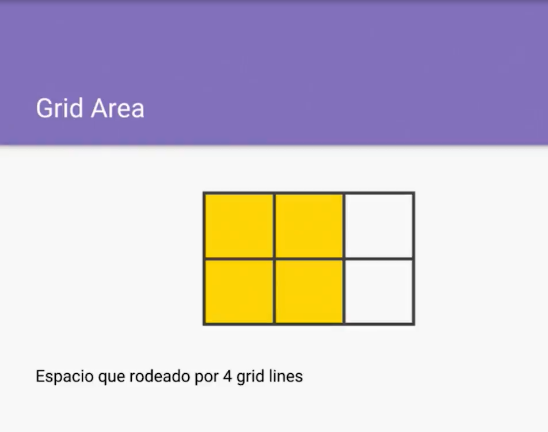
**Grid Track**: Espacio entre dos líneas adyacentes. Filas y columnas.



**Grid Cell**: Celdas, espacio en dos filas adyacentes y 2 columnas adyacentes.

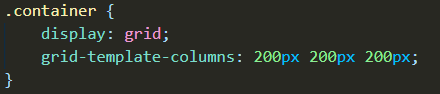


**Grid Area**: Espacio rodeado por 4 grid lines

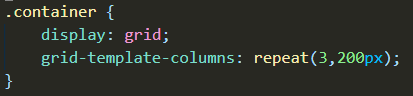


# Definiendo columnas.

# Para definir columnas, luego de asignarle *Display: Grid* a nuestro contenedor solo es cuestión de usar la propiedad *grid-template-columns: 200px 200px 200px*

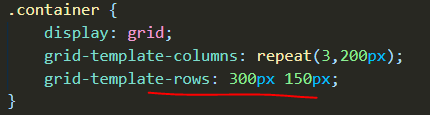


Otra forma de realizarlo es:

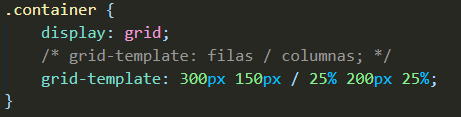


# Definiendo filas.

Para definir filas solo basta con usar la propiedad *grid-template-rows:* y asignaremos al igual que las columnas las medidas de la cantidad de filas que va tener nuestro grid layout.

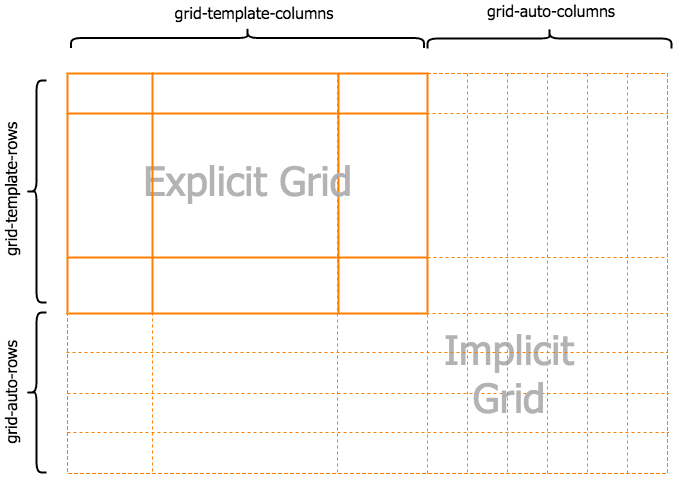


Sin embargo, también tendremos la posibilidad de asignar nuestras filas y columnas en una sola propiedad *grid-template* quedando de la siguiente forma:



**Grid explicito** (explicit grid) es cuando nosotros definimos el número de filas o columnas.

**Grid implícito** (implicit grid) es cuando tenemos filas o columnas que no definimos, pero son parte de nuestro grid.



# Grids identados y tipos de displays.

Para tomar un elemento de un tipo, o de una clase, le ponemos la pseudo clase *nth-of-type*(*número del elemento*), esto lo hacemos con el fin de darle estilos independientemente a el elemento que seleccionemos.



Para poder hacer un scroll dentro del elemento, debemos ponerle la propiedad *overflow* con el valor auto.



**Displays existentes con grid pero que actualmente no están soportados (24/09/2020):**

Display *subgrid* para heredar la configuración del grid padre (cuando se estén anidando grids).



Display *inline-grid* muestra el grid en una sola línea:



# Espaciado entre filas y columnas.

# Para añadir espaciado en nuestros elementos dentro de un grid las propiedades que nos ayudarían a realizarlo serian:

# 

# Repetidores, unidades de medida y funciones.

# fr = fracción:

# Es una unidad de medida que establece una división del layout en partes iguales; de manera homogénea entre la distribución de los ítems del contenedor.

# .

# Auto:

# El valor “auto” no queda cómo espacios iguales, ya que el tamaño varía según el contenido del elemento al que se asigna este valor.

# 

# Función repeat()

# Esta función nos permite repetir la cantidad de veces que le indiquemos la configuración de la unidad. *repeat(cantidad, valor)*

# .

# Función minmax()

# Esta función nos permite mantener un tamaño máximo y mínimo del elemento que estemos configurando.

# 

# Definiendo áreas de contenido.

# Para definir areas del maquetado utilizaremos la propiedad de grid: *grid-template-areas*: “header” “left contenido” “footer” (Pero si queremos especificar varias columnas debemos en nuestro ejemplo debemos escribir dos veces header y footer)

# 

# Para usar estas áreas:

# 

# Ejemplo:

# 

# 

# Definiendo el tamaño de las columnas dentro de un grid.

# Para definir el tamaño de las columnas dentro de un grid hago uso de las propiedades *grid-column-start:* y *grid-column-end.* Para definir debo tener en cuenta de que grid-line a que grid-line va ir el tamaño de mi columna.

# 

# En este caso es un grid de 3 columnas

# 

# De igual manera puedo realizar la misma asignación, pero solo con la propiedad *grid-column.*

# 

# En este caso tenemos un grid de 4 columnas.

# 

# *Span*

# También podemos hacer uso de *span* que nos dice cuántas cuadrículas ocupará a partir del inicio dado.

# 

# En este caso tenemos un grid de 4 columnas.

# 

# Cuando queremos que una columna ocupe todo el ancho podemos asignar la propiedad *grid-column* de la siguiente manera:

# 

# 

# Definiendo el tamaño de filas dentro de un grid.

# Para definir el tamaño de las filas se encontrarán disponibles las mismas propiedades que usamos para definir columnas, pero en este caso para filas(row): *grid-row.*

# 

# 

# 

# Definiendo el nombre de líneas.

# Para nombrar líneas colocamos el nombre entre [ ].

# 

# En estos casos no podemos usar la función repeat()

# Manejando el grid implícito.

# Cuando columnas o filas no tienen un valor explicito, es decir, cuando no asignamos valores específicos para controlarlas, se ordenan de manera implícita, hay propiedades que existen para manipular nuestro grid implícito:

# *grid-auto-flow:* Con esta propiedad manipulamos el orden del grid implícito. Si ingresamos el valor column, todo lo que sea grid implícito se apilara como columna y si ingresamos row, como fila.

# 

# *Por defecto viene: grid-auto-flow: row;*

# *grid-auto-column:* Con esta propiedad podemos darle valores a cada columna.

# *grid-auto-row:* Con esta propiedad podemos darle valores a cada fila.

# 

# Alineación de contenido.

***justify-items (Horizontal)***



VALUES:  
start : contenido hacia la izquierda  
end : contenido hacia la derecha  
center : contenido al medio  
stretch : estira el contenido al espacio que nos dé el grid. (**Valor por defecto**)

# *align-items (Vertical)*

# 

VALUES:  
start : contenido hacia arriba   
end : contenido hacia abajo  
center : contenido al medio  
stretch : estira el contenido al espacio que nos dé el grid. (**Valor por defecto**)

# *Alineación individual*

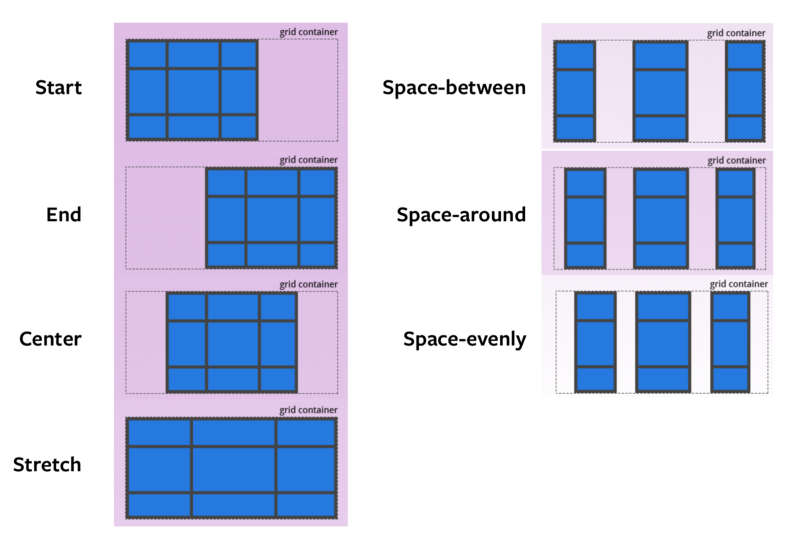
# justify-self (HORIZONTAL)

# align-self (VERTICAL )

# Alineación de filas y columnas.

Para alinear el contenido de filas y columnas:  
justify-content: valor; horizontal  
align-content: valor; vertical

Pata distribuir uniformemente:  
Con space-around Los ítems tienen el mismo espacio a su alrededor  
Con space-evenly Hay un espacio más homogéneo entre ítems  
Con space-between El primer ítem al inicio, el último al final



**Notas del ejercicio de Layout Mansory**

* Si se desea configurar los elementos grid para que continúen (o empiece) inmediatamente después del otro elemento se declara solo el grid-row-end seguido de la cantidad de espacios a ocupar.

*grid-row-end: span 3;*

* Para que las columnas se distribuyan dependiendo de la cantidad de espacio disponible en el viewport se usa el atributo auto-fill de la propiedad repeat (que sería más o menos igual a lo que hace flex con el wrap

.padre {

            grid-template-columns: repeat(auto-fill, 250px);

            }

NOTA: Esta forma de distribución llenara la pantalla de tantas columnas quepan en el viewport (así estén vacías)

* En caso de que quedaran espacios vacíos o no se ajustara los elementos correctamente al layout deseado, se puede usar *grid-auto-flow*, el cual es una propiedad que modifica la colocación automática de cada elemento en el grid. Su valor por defecto es el “sparse” el cual indica que si un elemento no  
  cabe en el “espacio” que dejo otro elemento más grande para colocarlo este saltara hasta encontrar un espacio  
  donde quepa el elemento siguiente.  
  Sin embargo con la propiedad “dense” esto se puede alterar, ya que con esta propiedad se indica que  
  en el espacio disponible coloque cualquier elemento que quepa en dicho espacio sin importar su orden.  
  NOTA: En el Dense se necesita declarar medidas de alto y ancho a los contenedores .hijos (al menos 100% para cada una)

**ULTIMAS CONSIDERACIONES**

**Extension Autocompletition CSS VsCode.**

[*https://marketplace.visualstudio.com/items?itemName=solnurkarim.html-to-css-autocompletion*](https://marketplace.visualstudio.com/items?itemName=solnurkarim.html-to-css-autocompletion)

*Para tener en cuenta:*

* Algunas etiquetas como <body> contienen estilos predeterminados. Debemos realizar un formateo de estilos inicial.
* Si quiero crear un botón sin funcionalidad puedo hacer uso de **<a>**
* Si mi botón debe tener funcionalidad (como en un form), puedo hacer uso de **<button>**
* Atajos para crear estructura HTML: **html5 + TAB** -------- **¡ + TAB**
* Crear texto dummy : **Lorem + TAB** ---- Lorem20 ----Lorem5----Lorem50
* Para crear línea de **subrayado** puedo hacer uso de la propiedad:

*border-bottom: 1px solid grey*

* Para crear una coherencia entre tamaños de elemento puedo definir width/height estáticos a los elementos padre y a los elementos hijos medidas relativas.
  + .padre {min-height: 250px;}
  + .hijo {width: 50%;}

***Documentación importante de Grid:***

[*https://css-tricks.com/snippets/css/complete-guide-grid/*](https://css-tricks.com/snippets/css/complete-guide-grid/)

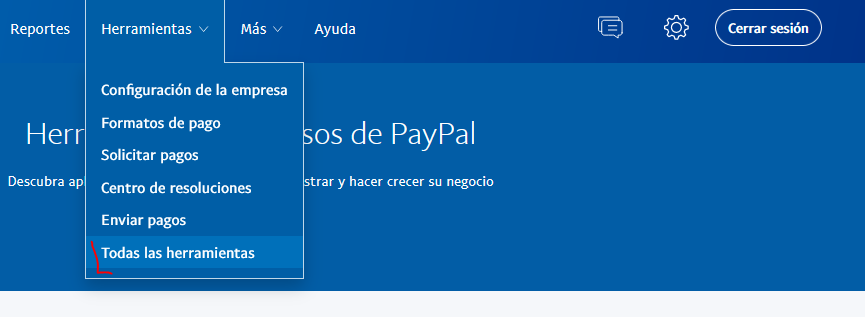
**Pagina para practicar código.**

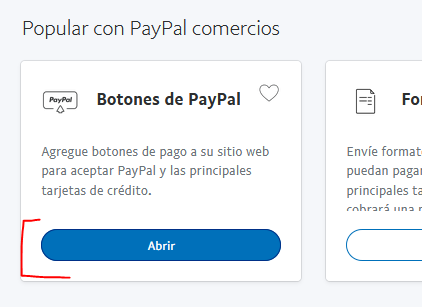
*https://www.codewars.com/*

# Configurar botón de pago PayPal a mi sitio web.

El botón de pago es una herramienta de PayPal que ayuda a configurar la cantidad y el método de pago para que nuestro usuarios paguen por nuestros productos o servicios de forma fácil y rápida.

Cuando configuremos nuestro botón, vamos a copiar la URL que generamos con PayPal y la vamos a pegar en el atributo action de nuestro formulario, además debemos añadir el atributo method="POST" para que nuestro formulario funcione correctamente.







[**https://uxplanet.org/the-sad-state-of-payment-buttons-4d5cef3b9578**](https://uxplanet.org/the-sad-state-of-payment-buttons-4d5cef3b9578)

[**https://medium.com/@bluepnume/less-is-more-reducing-thousands-of-paypal-buttons-into-a-single-iframe-using-xcomponent-d902d71d8875**](https://medium.com/@bluepnume/less-is-more-reducing-thousands-of-paypal-buttons-into-a-single-iframe-using-xcomponent-d902d71d8875)

[**https://www.paypal.com/us/webapps/mpp/logos-buttons**](https://www.paypal.com/us/webapps/mpp/logos-buttons)

[**https://writingcooperative.com/paypal-donations-vs-paypal-me-option-the-beginners-guide-to-medium-6-a2d35a9c8fe1**](https://writingcooperative.com/paypal-donations-vs-paypal-me-option-the-beginners-guide-to-medium-6-a2d35a9c8fe1)

[**https://medium.muz.li/these-11-tips-will-help-you-create-succesful-checkouts-b78f54a6751b**](https://medium.muz.li/these-11-tips-will-help-you-create-succesful-checkouts-b78f54a6751b)

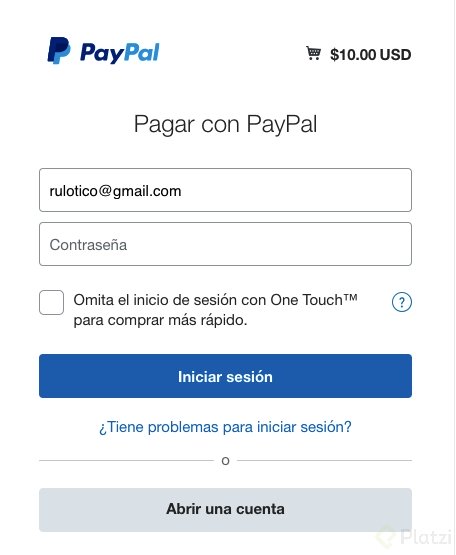
# Autofill del formulario de PayPal

Qué bueno que has llegado hasta aquí. En esta clase aprenderemos algunas cosas extras pero muy importantes sobre PAYPAL y el formulario que acabamos de hacer, lo más importante ahora es que seguro tienes dudas y una de ellas puede ser ¿qué pasa con las variables que envías por el formulario?¿cómo las recibe Paypal en su sistema y para qué nos sirve?

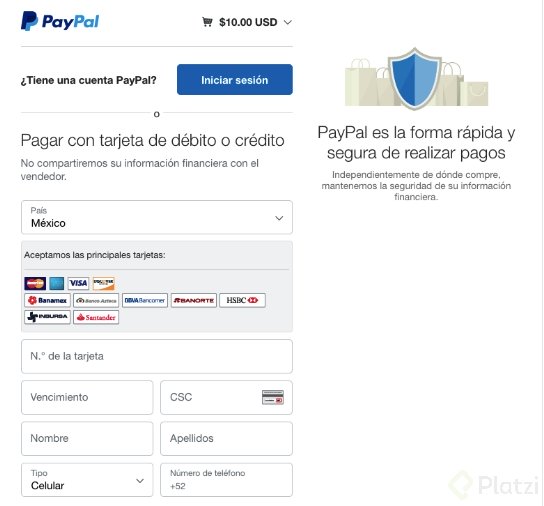
Vamos por pasos:

**Variables**

Las variables que nosotros enviamos en el código tal como name, phone e email Paypal las recibe para poder completar su formulario por ejemplo cuando nosotros no enviamos ninguna variable en el POST del formulario, la pantalla de Paypal será:

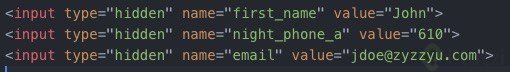


Al recibir data del formulario de PayPal cambia la pantalla por algo como esto:



Esto sucede porque las variables que enviamos por el formulario, después viajan por el URL y llegan a PayPal, son las mismas variables que utiliza para llenar su formulario y así el usuario no tiene que ingresar 2 veces la misma información.

Tu código debe tener estas variables para que la opción de autollenado funcione



Con esto puedes notar que tienes muchas opciones para tu formulario lo importante es que sigas el mismo formato que tiene paypal para pasar las variables.

Te dejo también un ejemplo más amplio de variables para los forms:

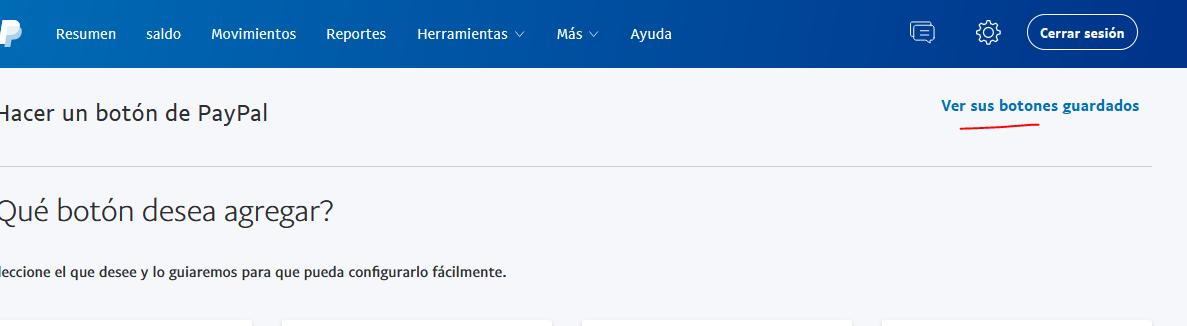


Toda la documentación de la función de autollenado la encuentras en este enlace:  
<https://developer.paypal.com/docs/classic/paypal-payments-standard/integration-guide/formbasics/#auto-fill-forms-with-html-variables>

Y las variables que pueden enviar para el auto llenado del formulario son :  
<https://developer.paypal.com/docs/classic/paypal-payments-standard/integration-guide/Appx_websitestandard_htmlvariables/#paypal-checkout-page-variables>

***Configurar página de éxito y falla en el pago.***

*Ingresamos a la sección de mis botones guardados y editamos nuevamente nuestro botón Paypal.*





# Crear un servidor de archivos estáticos con Node

Por ahora, aprenderás a realizar un servidor de archivos estáticos con Node, esto te permitirá contar con las herramientas necesarias para trabajar sobre el remote debugging en distintos dispositivos. Así que, en primer lugar, vas a descargar el software de Node, que te permitirá crear los archivos estáticos.

De esta manera podemos emular la pantalla de un dispositivo móvil en un servidor para archivos estáticos con node.js ya que no estamos manipulando un celular, sino un tamaño desde el navegador, por lo que debemos dirigirnos a la página principal de Node:

[**https://nodejs.org/en/**](https://nodejs.org/en/)

Bajar con el scroll luego y descargar la versión para nuestro navegador.

Luego de haber instalado node, debemos ejecutar la terminal de comandos de nuestro SO. Debemos verificar que npm está corriendo correctamente. NPM es el gestor de módulos de paquetes de node, los paquetes pueden ser cualquier código de javascript que hayamos escrito.

Debemos descargar un módulo ya creado por la comunidad que servirá de servidor de archivos estáticos. Para ello debemos escribir en nuestra terminal lo siguiente:

**npm -g install static-server**

Luego de ello debemos ubicarnos a la carpeta de nuestro proyecto, para saber en qué carpeta estamos ubicados en Windows es el comando dir, y para movernos entre carpetas nos movemos con el comando cd.

Una vez estemos ubicados en la carpeta, ejecutamos el comando “static-server”, una vez haya creado el servidor estático, tenemos los privilegios de detener la ejecución del servidor oprimiendo la combinación de **Ctrl + c**, pero en vez de eso, copiamos el puerto que nos arrojó (9080) por defecto, copiamos la url y podemos acceder a nuestro proyecto a través de esa url.

Luego de ello podemos acceder a cualquier otro dispositivo y observar nuestro proyecto. Para ello debemos verificar nuestra dirección ip con ipconfig desde la terminal (Windows).

**Ejemplo: 192.168.0.19:9080**

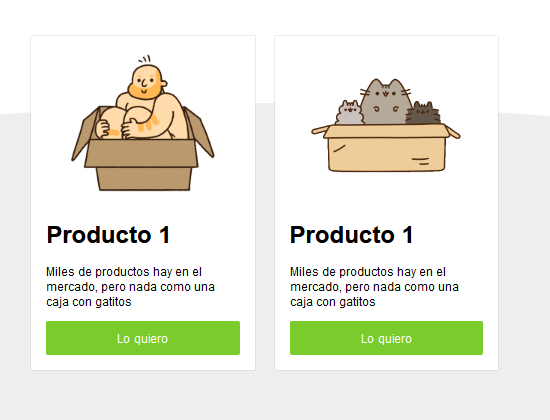
Nota: debemos tener un archivo index.html

# Maquetación de tarjetas de producto

Vamos a utilizar el atributo margin-top de CSS para darle espacio a nuestros elementos de producto, entre más grande sea el valor que configuremos en este atributo, más grande será el espaciado entre estos elementos y los de arriba, además, podemos utilizar valores negativos (margin-top: -50px) para disminuir este espaciado y conseguir el efecto que diseñamos.

También vamos a añadir sombras a nuestros elementos utilizando el atributo box-shadow: 0 0 2px black; de CSS. Si quieres configurar el color y dirección de estas sombras, puedes hacer click derecho en tus elementos, seleccionar ““Inspector de Elementos”” y configurar las sombras como más te gusten, recuerda copiar y pegar tus modificaciones porque, por defecto, no se guardan cuando refresques la página.

***Example:***

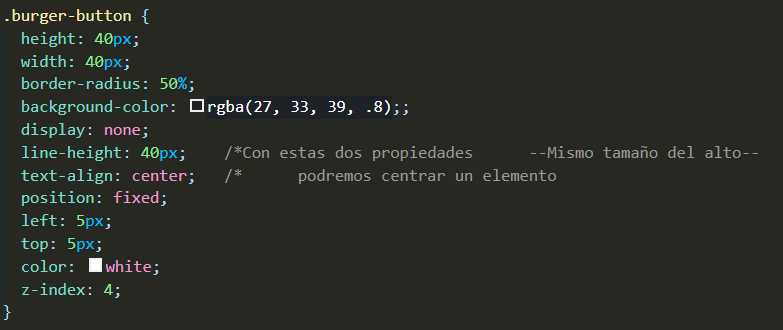


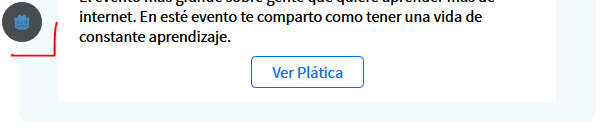
# Menú de hamburguesa

En esta clase convertirás el ícono que has elegido según las fuentes de íconos y le destinarás una función, es decir, le adjudicarás un “call to action” o llamado a la acción. Así, tu usuario sabrá cómo relacionar el ícono a la función que necesita realizar.

Para ello le añadiremos una nueva clase *burger-button* ajustándolo como un botón.



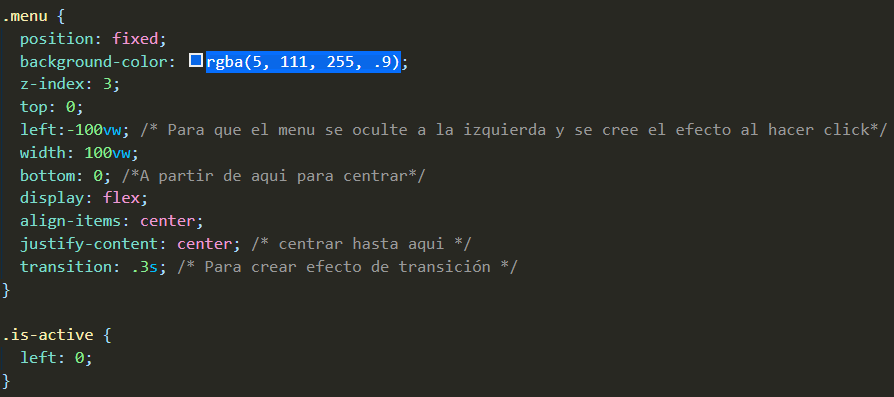




# Posicionando el menú

Antes de que tu menú tenga elementos que permitan la interactividad, es necesario definir sus posiciones y ordenamientos desde tu editor de código. En esta clase definirás los tamaños de tu menú, de los textos y, especialmente, trabajarás sobre sus estilos. Estos estilos los irás comprobando en distintas opciones de visualización o tamaños de dispositivos, con el propósito de estar realizando un trabajo efectivo en cuanto al responsive design. No obstante, al finalizar verás que aún no tiene la posibilidad de aparecer y esconderse, según lo necesite tu usuario, por este motivo: te invitamos a ver la próxima clase en donde usarás elementos de javascript para brindarle otra experiencia, mucho más completa, a tus usuarios.

* Primero agregaremos estilos a nuestro *.menu* que en este caso es nuestro navBar.



* Aquí añadimos una clase *.is-active* que nos permite mostrar el navBar ya que si nos fijamos, en los estilos nuestro *.menu* tiene un *left:-100vw;* y al colocarle un *left:0* lo que haría es que se visualice en nuestro viewport.
* Luego tendremos que darle interacción con JavaScript añadiendo una etiqueta <script> en nuestro HTML.

