

**CSS**

-Se puede dar estilos por etiqueta, class o id

Etiqueta= img, body, h1{}

Class= .nombreClase{}

Id= #nombreId{}

border-radius: 9px;

    box-shadow: 1px 1px 10px rgb(168, 135, 167), 0px 0px 25px 0px rgb(255, 182, 251);

    width:150px;

    height:100px;

**PSEUDOCLASES**

BEM (para nombrar clases en CSS)

Se asigna clase al contenedor que contenga las etiquetas que se requiere dar estilo

[Metodología \*\*BEM](https://www.youtube.com/watch?v=wDUwGo98JTA&feature=youtu.be) cómo nombrar correctamente las clases.

Solo usa **clases**. Las nombra siguiendo el siguiente patrón: **BLOQUE\_\_ELEMENTO—MODIFICADOR** (son 2 guiones, se usa 1 guión para separar palabras).

* **Bloque:** sección que puede funcionar por sí sola.
* **Elemento:** forma una de las partes del bloque.
* **Modificador:** variaciones de un mismo bloque/elemento.

Ejemplo BEM:

<header>

        <nav>

            <ul *class*="main-nav">

                <li *class*="main-nav\_\_item"><a *href*="#">Home</a></li>

                <li *class*="main-nav\_\_item"><a *href*="#">Cursos</a></li>

                <li *class*="main-nav\_\_item"><a *href*="#">Instructores</a></li>

                <li *class*="main-nav\_\_item"><a *href*="#">Blog</a></li>

            </ul>

        </nav>

    </header>

**PSEUDOCLASES Y PSEUDOELEMENTOS**

[**https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes**](https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes)

[**https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-elements**](https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-elements)

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

youtube- [BEM: Simplifica tu CSS](https://youtu.be/wDUwGo98JTA)

youtube- [Que rayos es la Metodología BEM](https://youtu.be/bvnzyXGkNY4)

Documentacion- [sobre BEM](http://getbem.com/introduction/)

Documentación- [Las Pseudo-classes hacen uso de un ‘:’](https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes)

Documentacion- [Los Pseudoelementos hacen uso de dos ‘::’](https://developer.mozilla.org/es/docs/Web/CSS/Pseudoelementos)

.main-nav{

*/\* Margen superior \*/*

    margin-top: 10px;

*/\* Eliminar puntos que aparecen en listas desordenadas \*/*

    list-style: none;

*/\* Elimina espacio a la izquierda dentro de la caja \*/*

    padding-left: 0;

    background-color: #13a4a4;

}

.main-nav\_\_item{

*/\* Para que el listado quede horizontal \*/*

    display: inline-block;

}

.main-nav\_\_item a{

    color: white;

*/\* Espacio entre los elementos \*/*

    padding: 5px;

*/\* Le agrega borde \*/*

    border-radius: 2px;

*/\* Elimina subrayado que genera la etiqueta "a" \*/*

    text-decoration: none;

}

*/\* PSEUDOCLASE \*/*

*/\* Efecto de oscurecer texto cuando se pasa el cursor por encima \*/*

.main-nav\_\_item a:hover{

    color: rgba(255, 255, 255, 0.5);

}

*/\* Cambia el color del elemento al cual se dió click \*/*

.main-nav\_\_item a:active{

    color: rgba(255, 255, 255, 0.5);

}

*/\* PSEUDOELEMENTOS \*/*

*/\* Agrega un caracter después del texto de la etiqueta \*/*

.main-nav\_\_item a::after{

    content: " | ";

}

**ANATOMIA DE UNA REGLA DE CSS**

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media**

**MODELO DE CAJA**

**Interfaz de usuario gráfica

Descripción generada automáticamente**

En la web los elementos se piensan en forma de rectángulos. A esto se le conoce como Modelo de caja.

**Margin**

Espacio que hay del borde de la caja hacia afuera. Visualmente, permite separar a la caja de elementos externos.

**Padding**

Espacio que hay del borde de la caja hacia adentro. Visualmente, permite separar a la caja de elementos internos.

**Border**

Es el borde de la caja. Por defecto, a partir de ahí medimos las distancias de nuestra caja con elementos internos o externos.

Imagen que contiene Código QR

Descripción generada automáticamente

**Width**

Largo del contenido

**Height**

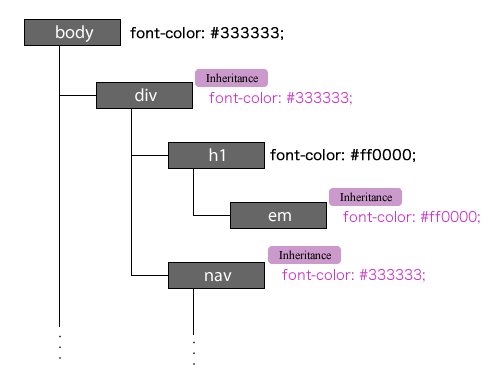
Alto del contenido

**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 ultima 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.

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



**ESPECIFIDAD EN SELECTORES**

**Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente**

**Tabla

Descripción generada automáticamente** **Diagrama

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

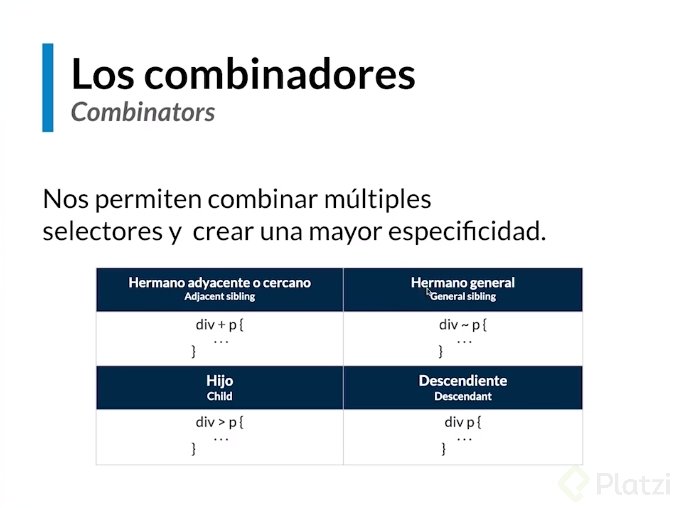
1. Los **!important** estarán por encima de los demás estilos. Sin embargo, son mala práctica y no se deberían usar.
2. Los estilos embebidos en el HTML, es decir **inline styles** están por encima de las **clases y IDs**. Sin embargo, también se deberían evitar.
3. Los **IDs** están por encima de las **clases**. Los **IDs** son específicos, si se usa uno en un archivo HTML ya no se podrá repetir más en ese mismo archivo. Mientras que las **clases** si se pueden repetir en cualquier elemento.
4. Un **estilo de etiqueta** es el último valor que el navegador tiene en cuenta antes de tomar los estilos por defecto de esa etiqueta. Los **estilos de etiqueta** son los que menos peso tienen.

**MALAS PRÁCTICAS DICHAS EN CLASE HASTA AHORA**

* Utilizar tanto **id** en CSS
* Utilizar el **!important**
* Utilizar la etiqueta **<style>** dentro del archivo html
* Utilizar el atributo style dentro de las etiquetas html
* Utilizar div para contener todo ignorando los header, nav, section, article, etc.
* No utilizar la etiqueta **<form>** para hacer formularios
* Utilizar las etiquetas <select> y <option> para hacer selectores o menús desplegables.
* No nombrar el primer archivo html del proyecto como index.html
* No tener archivos .css para cada pantalla de un proyecto.
* Tener todo el css junto en un solo archivo.
* No ponerle el atributo alt a una imagen
* Poner imágenes dentro de <div> en vez de **<figure>**
* Utilizar textos solo en mayúscula en HTML, en vez de utilizar el atributo de CSS, text-transform, con el valor uppercase. Ya que al hacer esto pareciera que estuvieras gritando.
* Poner videos que se reproduzcan solos.
* No optimizar las imágenes.
* No tener cuidado de cual es el formato ideal para las imágenes y su respectivo peso.
* No tener cuidado con la respectiva semántica de HTML, y con la sintaxis adecuada para CSS.
* No cerrar las etiquetas que se cierran en sí mismas como <br/>
* No comentar partes esenciales de tu código.
* No poner la etiqueta **<meta name=”robots” content=”index,follow”>** en tu proyecto para que los navegadores los puedan ubicar mejor.
* No usar la etiqueta **<meta name=”viewpor” content=”width=device-width, initial-scale=1.0”>** para hacer tu proyecto responsive.
* No poner el atributo **autocomplete=”valor”** en los campos de tu formulario para hacerle la vida más fácil al usuario
* No usar el atributo **required** en los campos obligatorios de tu formulario como una primera capa de seguridad

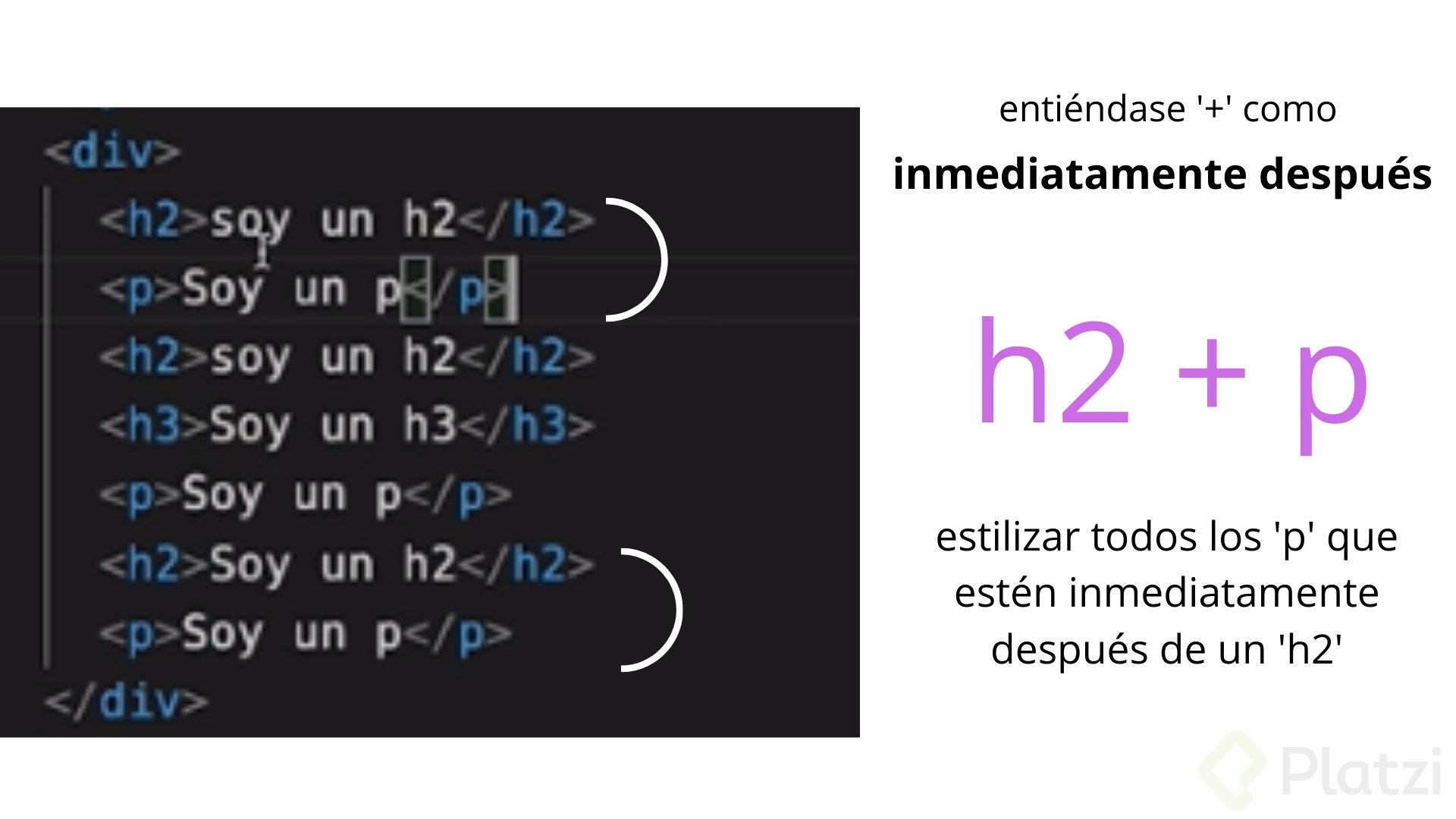
1.- Busca siempre una manera de emplear la especificidad antes de considerar el uso de !important.  
2 .- Usa !important solo en declaraciones específicas de CSS que sobrescriban CSS externo (de librerías externas como Bootstrap o normalize.css).  
3.- Nunca uses !important cuando estés intentando escribir un plugin/mashup.  
4.- Nunca uses !important en todo el código CSS.

**COMBINADORES: ADJACENT SIBLINGS (COMBINATORS)**



**HERMANO ADYACENTE O CERCANO**

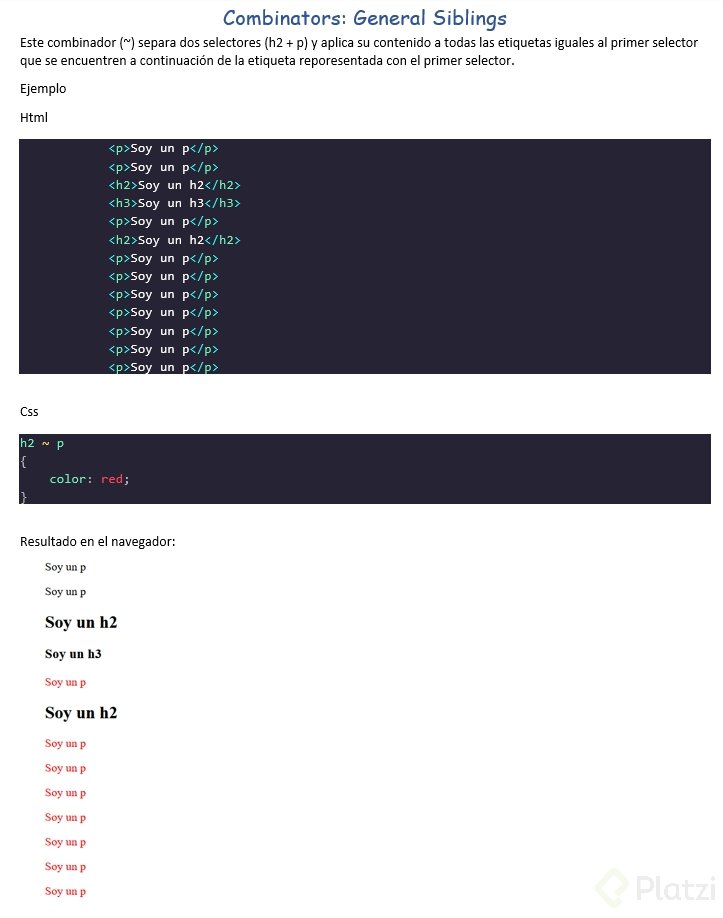
**ADJACENT SIBLING**



**GENERAL SIBLING**

es el combinador de **Hermano General**.  
Este tipo de combinador nos ayuda a seleccionar a los hermanos generales, es decir a los que comparten un mismo padre. Se representa mediante el siguiente simbolo \*\* ~ \*\* y se implementa asi.

**Codigo HTML**



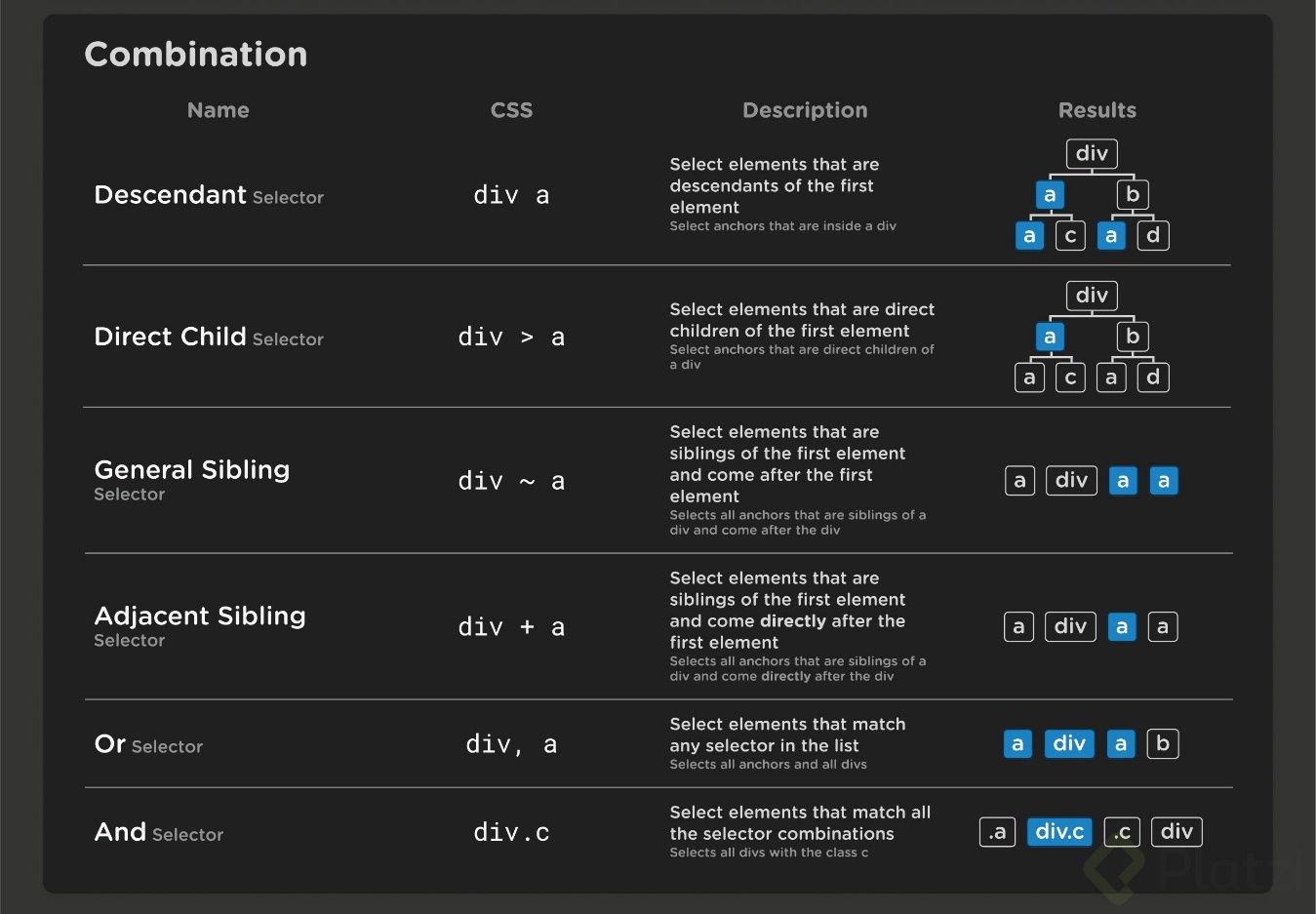
**COMBINADORES HIJO Y DESCENDIENTE**

**Interfaz de usuario gráfica

Descripción generada automáticamente con confianza baja**

**HIJO:** Etiqueta P que sea hija directa de etiqueta DIV

**DESCENDIENTE:** Todas las etiquetas P contenidas dentro del DIV principal



**MEDIDAS**

Tabla

Descripción generada automáticamente

medida absoluta: el valor de este no cambia y siempre será el mismo así la página cambie su tamaño, las medidas absolutas son:  
.  
mm = milimetros  
cm = centimetros  
in = pulgada  
pc = picas  
px = pixel  
.

las medidas relativas: estas medidas heredan el tamaño o se basan en algún tamaño que se allá seleccionado y el valor irá cambiando según si la página cambia de tamaño, las medidas relativas son :  
.  
%  
em  
rem

**MEDIDAS EM**

es un acronimo de elemento y lo que hace es tomar el tamaño de fuente que tenga el padre directo ejemplo:

.container {

font-size: 20px

}

.container **div** {

font-size: 2em

}

aqui los el tamaño del div que esta dentro de la clase container tenda un tamaño de 40px, ya que  
.  
1em = 20px  
.  
y como estamos asignandole 2 em seria como 20px + 20px  
.  
y si por ejemplo tenemos el siguiente caso :

.container {

font-size: 20px

}

.container **div** {

font-size: 2em

}

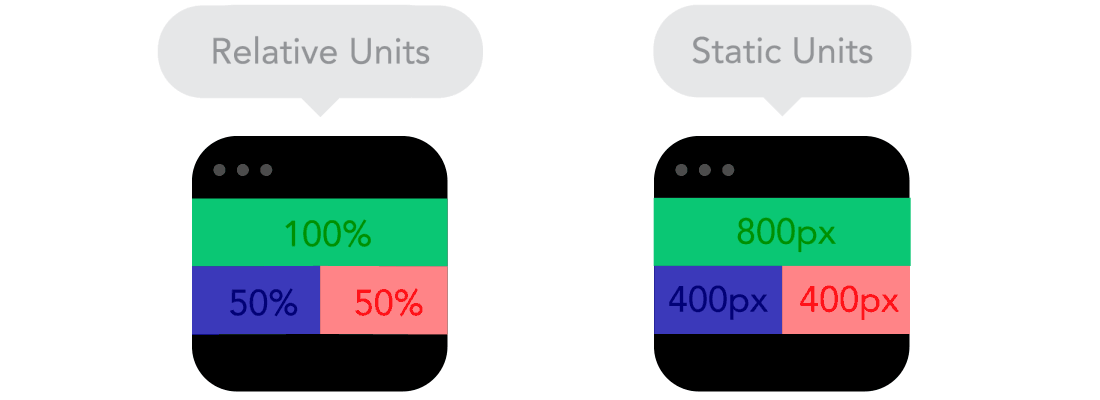
.container **div** **p** {

font-size: 1.5em

}

a continuacion veremos que la etiqueta p tendra un tamaño de fuente de 60px  
ya que toma como referencia el tamaño de su padre ( 40px ) y haria la siguiente operacion 40\*1.5 que es igual a 60, es por eso que la etiqueta p tomo el valor de 60px

**REM**



*/\* Siempre debe ir en los archivos de CSS \*/*

\* {

    box-sizing: border-box;

    margin: 0;

    padding: 0;

}

html {

    font-size: 62.5%;

}

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 asi que siempre  
.  
1 REM = 16PX  
.  
Si queremos aplicar rem de una forma mas sencilla para no tener que hacer tantos calculos asemos lo siguiente  
.  
vamos a reescribir en css nuestro html

**html** {

font-size: 62.5%;

}

esto lo que hara es darle a el html un valor de 10px ya que 16px - 62.5% = 10px

ahora si por ejemplo a una etiqueta le asignamos 2rem este hara referencia a 20px, o si por ejemplo le damos un valor de 1.5rem su valor sera de 15px

**MAX/MIN WIDTH**

**Min-height:** se utiliza para definir la altura mínima de un elemento dado. Impide que el valor de la propiedad height llegue a ser más pequeña que la especificada en la altura mínima ( min-height ). Se refiere a la altura del bloque contenedor.

**Max-height:** se utiliza para definir la altura máxima de un elemento dado. Impide que el valor de la altura pueda llegar a ser más grande que la de max-height . Porcentajes: se refiere a la altura del bloque contenedor.

**Min-width:** se usa para determinar la anchura mínima de un elemento. Previene que la propiedad width pueda ser inferior que min-width . Aplicable a: elementos de tipo bloque. Porcentajes: se refieren a la anchura del bloque contenedor.

**Max-width:** define el ancho máximo que un elemento puede tener, max-width cambia el tamaño del elemento si el valor de width es mayor que el de max-width.

**POSITION**

**Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente**

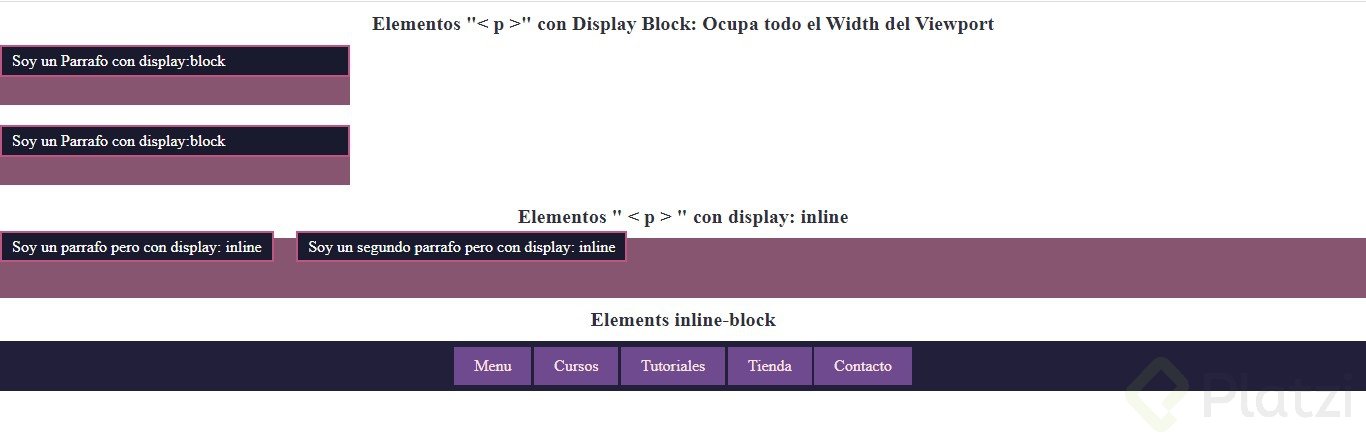
**DISPLAY**

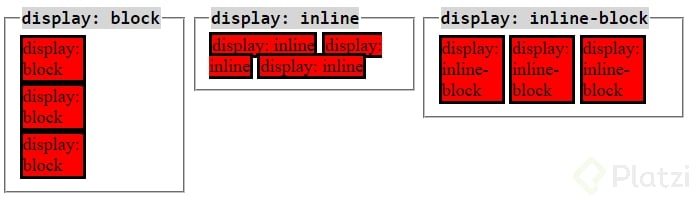
En esta clase vimos los 3 tipos de display Base.  
Los cuales fueron:

**Block**: Estos toman el 100% del width, por lo que un elemento no puede posicionarse a un lado de el.  
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 asi decirlo.

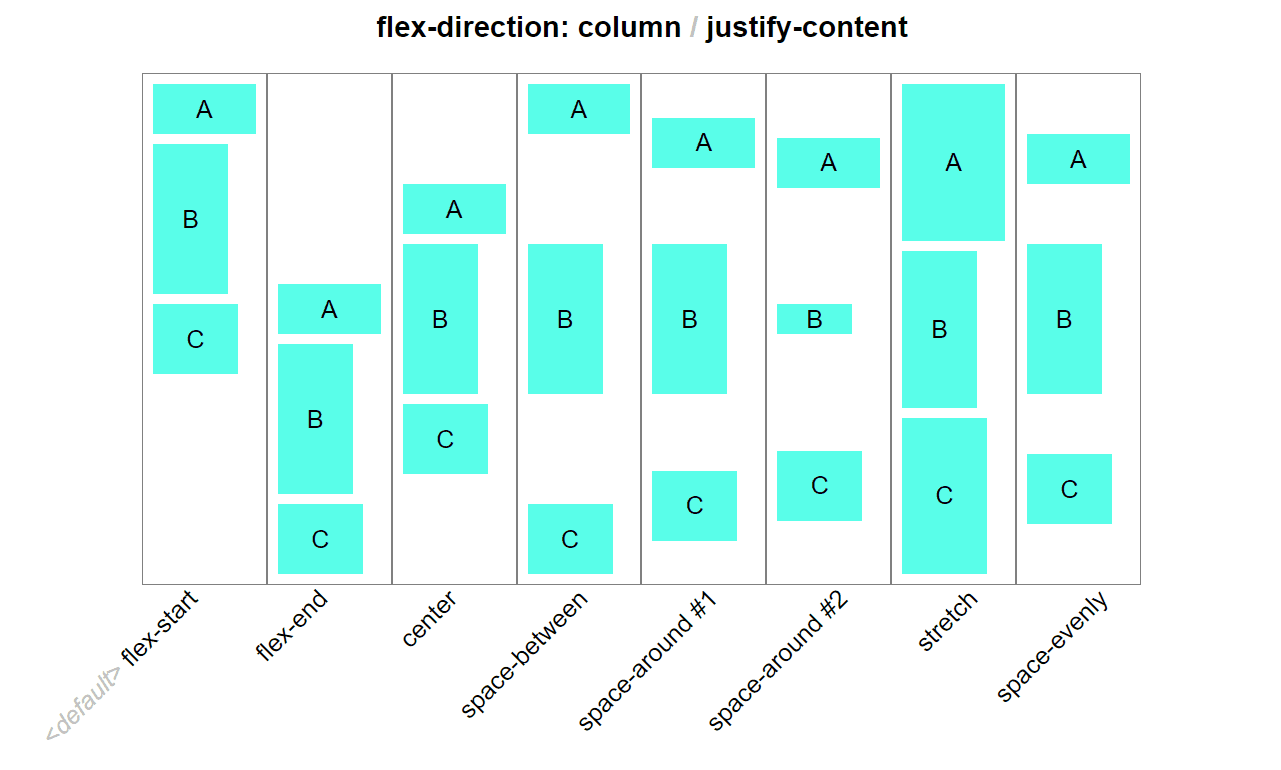
**Inline**: Estos elementos solo ocuparan el ancho dependiento de su contenido. Por lo tanto estos elementos si permiten que si un elemento cabe a lado suyo, se posicione este ahi.  
Las \**desventajas* es que no se les puede modificar el width, height, ni colocar margin u padding tanto top, como bottom.

**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 mas elementos ahi.





**DISPLAY FLEX**



***Mini guía de flexbox:***

**Propiedades en contenedores padre:**  
display:flex;  
flex-direccion: row | column | row-reverse | column-reverse  
flex-wrap: nowrap | wrap | wrap-reverse  
***Esta siguiente propiedad es un atajo para escribir el flex-direccion y el flex-wrap en una sola línea de código***  
flex-flow: *Primero escribes dirección* | *Luego escribes flex-wrap*  
***Posicionar horizontal***  
justify-content: flex-star | flex-end | center | space-around | space-between  
***Posicionar manera vertical***  
aling-items: flex-star | flex-end | center | stretch | baseline

aling-content: flex-star | flex-end | center | stretch | space-around | space-between “**Aling-conten solo se utiliza varias filas de elementos, pero si es una sola línea usamos aling-items**”

***Propiedades en elemento hijo***

order: ; Esto se utiliza para cambiar el orden de nuestros elementos sin cambiar el orden real semantico y correcto de html. Sencillamente colocando números.

aling-self: aling-items: flex-star | flex-end | center | stretch | baseline **“Muy importante, si en el padre del elemento tiene declarado flex-direccion:row; esta propiedad lo acomodara verticalmente. Y si es flex.direccion: column lo ordenara horizontalmente”**

**Flex** es un tipo de **display** que permite que el contenedor padre sea flexible a los cambios que puedan tener los elementos hijos en su alineación.

Una vez tengamos el elemento padre con display **flex** tenemos otras propiedades que podremos usar para nuestro beneficio.

* **Flex-direction:** Te permite elegir la alineación de los elementos hijos sea en vertical (*column*) u horizontal (*row*), esta alineación viene por defecto.
* **Justify-content:** Esta propiedad nos permite alinear el contenido de forma horizontal
  + Valores:
    - **Flex-start:** Alinear items del flex desde el comienzo.
    - **Flex-end:** Alinear items desde el final.
    - **Center:** Alinear items en el centro del contenedor.
    - **Space-between:** Distribuir *items* uniformemente, el primer *items* al inicio, el último al final.
    - **Space-around:** Distribuir items uniformemente, estos tienen el mismo espacio a su alrededor.
    - **Space-evenly:** Distribuye uniformemente el espacio entre los items y su alrededor.
* **Flex-wrap:** Permite que un elemento cuyo tamaño sea mayor al de la página haga un salto de línea sin salirse del contenedor, pues este se agranda.

**align-items:** Alinear de forma vertical

Esta propiedad CSS alinea elementos verticalmente y acepta los siguientes valores:

* flex-start: Alinea elementos a la parte superior del contenedor.
* flex-end: Alinea elementos a la parte inferior del contenedor.
* center: Alinea elementos en el centro (verticalmente hablando) del contenedor.
* baseline: Muestra elementos en la línea base del contenedor, se ajusta al espacio de su contenido
* stretch: Elementos se estiran para ajustarse al contenedor al 100%.

**justify-content:** Alinear de forma horizontal

propiedad **justify-content**, la cual alinea elementos horizontalmente y acepta los siguientes valores:

* **flex-start**: Alinea elementos al lado izquierdo del contenedor.
* **flex-end**: Alinea elementos al lado derecho del contenedor.
* **center**: Alinea elementos en el centro del contenedor.
* **space-between**: Muestra elementos con la misma distancia entre ellos.
* **space-around**: Muestra elementos con la misma separación alrededor de ellos.

**flex-direction:** Esta propiedad CSS define la dirección de los elementos en el contenedor, y acepta los siguientes valores:

* row: Elementos son colocados en la misma dirección del texto.
* row-reverse: Elementos son colocados en la dirección opuesta al texto.
* column: Elementos se colocan de arriba hacia abajo.
* column-reverse: Elementos se colocan de abajo hacia arriba.

**order**

A veces, invertir el orden de una fila o columna de un contenedor no es suficiente. En esos casos, nosotros podemos aplicar la propiedad **order** a elementos individuales. Por defecto, los elementos tienen un valor 0, pero nosotros podemos usar esta propiedad para establecerlo a un número entero positivo o negativo.

Usa la propiedad **order** para reordenar las ranas de acuerdo a sus hojas de lirio.

#pond {

display: flex;

}

.yellow {

order: 1

}

\*Si a algunos elemento no se agrega orden automáticamente se ubican al lado izquierdo y luego si inician los elementos ordenados

**align-self**

Otra propiedad que puedes aplicar a elementos individuales es **align-self**. Esta propiedad acepta los mismos valores de **align-items** y sus valores son usados para un elemento específico

**flex-wrap**

* nowrap: Cada elemento se ajusta en una sola línea.
* wrap: los elementos se envuelven alrededor de líneas adicionales.
* wrap-reverse: Los elementos se envuelven alrededor de líneas adicionales en reversa.

**flex-flow**

Las dos propiedades flex-direction y flex-wrap son usadas a menudo en conjunto con la propiedad abreviada flex-flow, la cual fue creada para combinarlas. Esta propiedad abreviada, acepta un valor de cada una separada por un espacio.

Por ejemplo, puedes usar flex-flow para establecer filas y envolverlas.

**flex-flow: wrap column**

Puedes usar align-content para establecer como múltiples líneas están separadas una de otra. Esta propiedad acepta los siguientes valores:

* flex-start: Las líneas se posicionan en la parte superior del contenedor.
* flex-end: Las líneas se posicionan en la parte inferior del contenedor.
* center: Las líneas se posicionan en el centro (verticalmente hablando) del contenedor.
* space-between: Las líneas se muestran con la misma distancia entre ellas.
* space-around: Las líneas se muestran con la misma separación alrededor de ellas.
* stretch: Las líneas se estiran para ajustarse al contenedor al 100%.

Esto puede ser confuso, pero align-content determina el espacio entre las líneas, mientras que align-items determina como los elementos en su conjunto están alineados dentro del contenedor. Cuando hay solo una línea, align-content no tiene efecto.

**FLEXBOX LAYOUT**

**Flex** es un tipo de *display* que permite que el contenedor padre sea flexible a los cambios que puedan tener los elementos hijos en su alineación.

Una vez tengamos el elemento padre con display: flex tenemos otras propiedades que podremos usar para nuestro beneficio.

* **Flex-direction:** Te permite elegir la alineación de los elementos hijos sea en vertical (*column*) u horizontal (*row*), esta alineación viene por defecto.
* **Justify-content:** Esta propiedad nos permite alinear el contenido de forma horizontal
  + Valores:
    - **Flex-start:** Alinear items del flex desde el comienzo.
    - **Flex-end:** Alinear items desde el final.
    - **Center:** Alinear items en el centro del contenedor.
    - **Space-between:** Distribuir items uniformemente, el primer items al inicio, el último al final.
    - **Space-around:** Distribuir items uniformemente, estos tienen el mismo espacio a su alrededor.
    - **Space-evenly:** Distribuye uniformemente el espacio entre los items y su alrededor.
* **Align-items:** Sirve para alinear los elementos hijos de forma vertical.
  + Valores:
    - **Flex-start:** Se alinean desde arriba.
    - **Flex-end:** Se alinean desde abajo.
    - **Center:** Alinea los item al centro del *eje y* del contenedor.
    - **Stretch:** Estira el alto de los elementos hijos al 100% del alto del elemento padre.
    - **Baseline:** Escala el alto del elemento al tamaño de su contenido.
* **Flex-wrap:** Permite que un elemento cuyo tamaño sea mayor al de la página haga un salto de línea sin salirse del contenedor, pues este se agranda.
* **Order:** Especifica el orden utilizado para disponer los elementos en su contenedor flexible. Los elementos estarán dispuestos en orden ascendente según el valor de order.
* **Flex-grow:** Especifica qué cantidad del espacio restante dentro del contenedor flexible, debería ocupar el ítem en cuestión según su dirección principal, esta última determinada por **flex-direction**.
* **Flex-basis:** Especifica el tamaño inicial de un elemento flexible.