**Curso de CSS**

* **Introduction to CSS**
* **CSS Essential Training 1-2-3**

Contenido

[Glosario 2](#_Toc1387086)

[Links útiles 4](#_Toc1387087)

[2.1 CSS Introducción 5](#_Toc1387088)

[Capítulo 1. Basics 5](#_Toc1387089)

[Capítulo 2. Specifications 9](#_Toc1387090)

[Capítulo 3. Common Concepts 9](#_Toc1387091)

[Capítulo 4. CSS Layout 15](#_Toc1387092)

[Capítulo 5. Working with CSS 15](#_Toc1387093)

[Capítulo 6. Responsive CSS 15](#_Toc1387094)

[Capítulo 7. Going further with CSS 16](#_Toc1387095)

[2.2 CSS Essential Training 1 17](#_Toc1387096)

[Capítulo 1. Getting Started 17](#_Toc1387097)

[Capítulo 2. CSS Core 18](#_Toc1387098)

[Capítulo 3. Typography 19](#_Toc1387099)

[Capítulo 4. Layouts 24](#_Toc1387100)

[2.3 CSS Essential Training 2 31](#_Toc1387101)

[Capítulo 1. CSS Selectors 31](#_Toc1387102)

[Capítulo 2. Layouts 36](#_Toc1387103)

[Capítulo 3. Tips and Tools 38](#_Toc1387104)

[Capítulo 4. Responsive design 40](#_Toc1387105)

[2.4 CSS Essential Training 3 42](#_Toc1387106)

[Capítulo 1. Layout 42](#_Toc1387107)

[Capitulo 2. Retina Display and Images 48](#_Toc1387108)

[Capítulo 3. Animaciones 48](#_Toc1387109)

[Capítulo 4. Tools and Tips 52](#_Toc1387110)

[2.5 CSS Grid 54](#_Toc1387111)

[Lección 01 – Práctico Introducción 54](#_Toc1387112)

[Lección 02 – Terminologia y primeros pasos 56](#_Toc1387113)

[Lección 03 – Separación entre los elementos Grid 60](#_Toc1387114)

[Lección 04 Y 05 – Usando Nombres GRID Lines START Y END 63](#_Toc1387115)

[Lección 06 – Unidad FR 65](#_Toc1387116)

[Lección 07 – GRID Template Areas 66](#_Toc1387117)

[Lección 08 – Establecer límites con minmax() 67](#_Toc1387118)

[Lección 09 – Repetir valores y patrones con repeat() 68](#_Toc1387119)

[Lección 10 – GRID dinámico con auto-fill y auto-fit 69](#_Toc1387120)

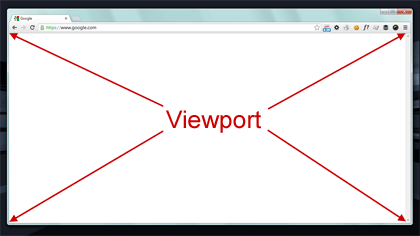
[Lección 11 y 12 – GRID Explicit e Implici, auto-placement y packing modes. 71](#_Toc1387121)

[Lección 13 – Re ordenar Grid ítems con la propiedad order 74](#_Toc1387122)

[Lección 14 75](#_Toc1387123)

# Glosario

* **Padding (to pad):** relleno/rellenar
* **Rule of thumb:** es una expresión del idioma inglés que designa un principio o criterio de amplia aplicación que no es necesariamente preciso ni fiable en toda situación
* **Viewport:** Lo que el usuario ve.



* **SEO:** Search engine optimization
* **DOM (Document Object Model):** Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento
* **CANVAS API:** El elemento <canvas> también se usa en [WebGL](https://developer.mozilla.org/es/docs/Web/WebGL) para dibujar gráficos 3D con aceleración por hardware en páginas web.
* **Vendor Prefixes*:***Permite a los manufactureros una forma de añadir soporte para nuevos features de CSS antes de que esos features sean totalmente soportados en todos los navegadores.
* **CDN:** Content Delivery Network
* **Retina:** Es un concepto de apple para las pantallas que tienen una densidad de pixeles mayor.
* **DPI:** Dots per inch. Unidad de densidad de pixeles.

# Links útiles

* (Explica todas las propiedades de CSS) <https://tympanus.net/codrops/css_reference/>
* <https://css-tricks.com/>
* <https://www.w3.org/Style/CSS/current-work.en.html>
* <https://www.w3schools.com/css/>
* Compatibilidad de elementos en los distintos navegadores <https://caniuse.com/>
* Calculadora de Especificidad <https://specificity.keegan.st/>
* Class naming methodologies: <https://webdevstudios.com/2017/03/28/evolution-css-class-naming-methodologies/>
* Guías para escribir bien CSS <http://codeguide.co/>

# 2.1 CSS Introducción

## Capítulo 1. Basics

3 formas de aplicar CSS a un HTML.

No se debe usar *inline styles* casi nunca y la forma más recomendada es de un archivo externo.

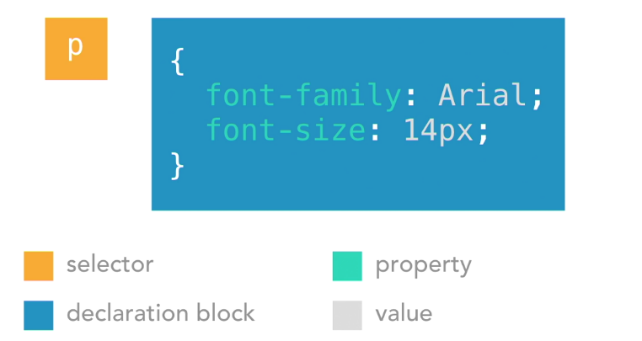


¿Cuál style.css tiene prioridad?

Los estilos más específicos son los que terminan tomando prioridad sobre los otros.



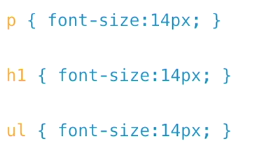
Partes del CSS

En la siguiente imagen se describe cada una de las partes de un bloque de CSS.

Selectores

Existen cuatro tipos de selectores para referenciar a los elementos de HTML:

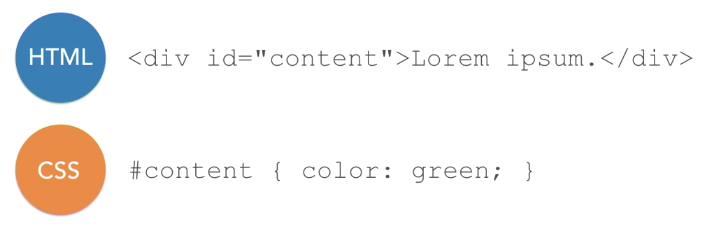
***Selector de un elemento****:* Elementos de HTML que al ser seleccionados son aplicados globalmente. En la imagen, todo los *p* tendrán un tamaño de 14px. Asimismo, con los otros dos elementos.



***Selector de clases****:* Atributos aplicados a un elemento de HTML. Se pueden usar varias veces con diferentes elementos

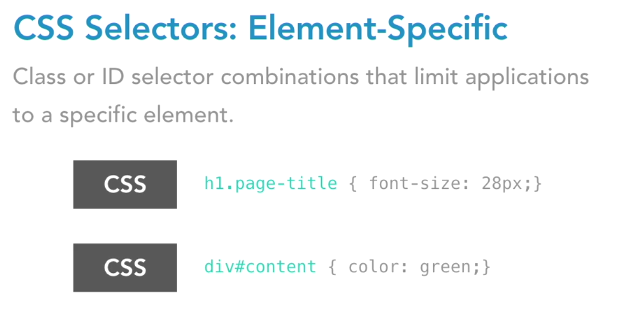


***Selector de ID****:* Representan un elemento de HTML, pero son únicos.



Elementos específicos combinando.

De esta forma se puede acceder a un elemento especifico requerido. Ya sea un *H1* con la clase de *page-title* como se ve en la imagen. Logrando discriminar a los otros elementos que posean esta clase o a los *H1* en general.



Selectores de descendientes:

También, se puede acceder a selectores descendientes, es decir que el navegador, en este ejemplo, buscara cada elemento *anchor* dentro del elemento *paragraph* que también se encuentre dentro de un *div.*



Agrupación de selectores:

Asimismo, para aplicar un mismo estilo a diferentes selectores, ya sean por clase, id, elemento, etc., se puede realizar simplemente separando por comas.



La última regla prevalece



Es decir que, el último estilo aplicado, es el que va a prevalecer.

## Capítulo 2. Specifications

CSS 3 – level three

Es la última evolución del lenguaje actual de CSS y pretende ampliar la versión de CSS2.1. Formalmente, no existe un estandar de CSS3 por sí solo cada módulo es estandarizado independientemente, por lo que el estandar CSS consiste en CSS2.1 modificado y extendido por módulos terminados, no necesariamente todos con el mismo nivel numérico

## Capítulo 3. Common Concepts

Reset y normalize

Esto se refiere a las técnicas para eliminar todo tipo de estilos incluidos por defectos en los navegadores. Sin embargo, estas dos técnicas tienen algunas diferencias:

* **Reset:** La técnica *reset.css* hace que todos los valores que tiene un navegador por defecto en cuanto a estilos vuelvan a su valor cero para lograr aplicarles un estilo a nuestro antojo. Es como tener un lienzo en blanco. El proyecto más famoso es el Reset de Eric Meyer.
* **Normalize:** Se le conoce como un *nuevo reset* y lo que logra es que, en vez de dejar los estilos en cero, se encarga de que los estilos tengan una base que sean iguales en todos los navegadores.

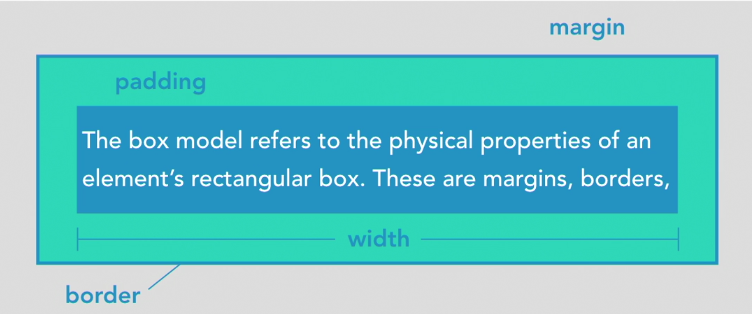
Working with fonts:

Algunas de las propiedades más comunes trabajando con fuentes son:

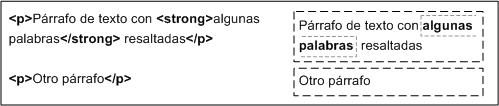
* Font-family
* Font-weight
* Font-style
* Font-size
* @font-face: Cargar fuentes de la web. Se cargan vía JavaScript.

The Box Model – Modelo de cajas

Se refiere a las propiedades físicas de un elemento. Es probablemente la caracteristica más importante de CSS. Hace que todos los elementos de las páginas se representen mediantes cajas rectangulares.

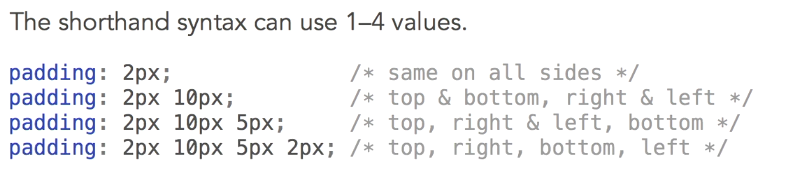


Las cajas se crean automaticamente. Cada vez que se inserta una etiqueta de HTML, se crea una caja rectangular que encierra el contenido de ese elemento, ejemplo:

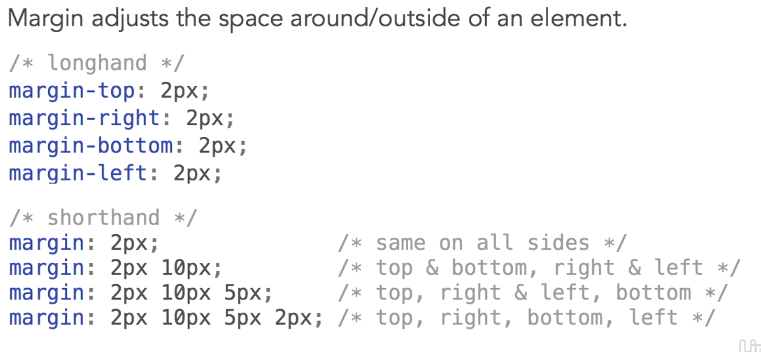


Elementos del Box Model – Modelo de cajas

* **Width and Height:** Son usados para cambiar el tamaño por defecto de elementos *Block* o *Inline.* Especifica el tamaño para el contenido de la caja.
* **Padding:** Ajustar el espacio entre el contenido y el borde. Es un “relleno”

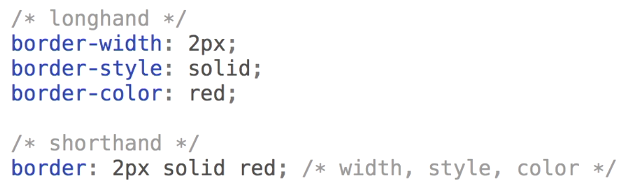


* **Margin:** Ajusta el espacio alrededor/afuera de un elemento, es decir entre la caja existente y el resto adyacentes.



Margin: 0 auto: ¡Logra centrar el contenido!

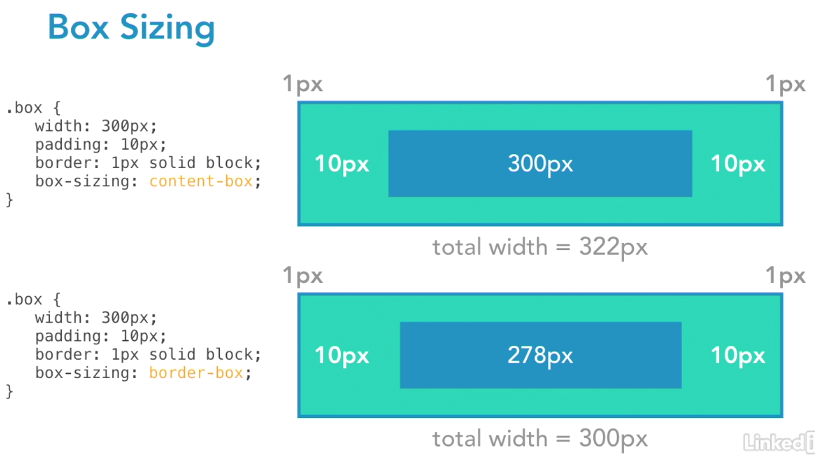
* **Border:** Es el borde entre el *margin* y el *padding.*



Box-sizing property

Esta propiedad permite indicar al agente la forma como se debe calcular el ancho y el alto de un elemento.

* **Content-box:** Es el comportamiento por defecto. Como se ve en la primera imagen el *width* define solo el ancho de elemento y cualquier relleno o borde se incluirá en la sumatoria total.
* **Border-box:** Por otra parte, este valor, como se muestra en la segunda imagen, indica que el *width* será de 300 px y los demás elementos se irán ajustando a este.



Padding and Margins

La sintaxis de ambos elementos es muy parecida. Los *Paddings* representan el relleno del contenedor y el *Margin* los espacios adicionales alrededor o afuera del elemento.

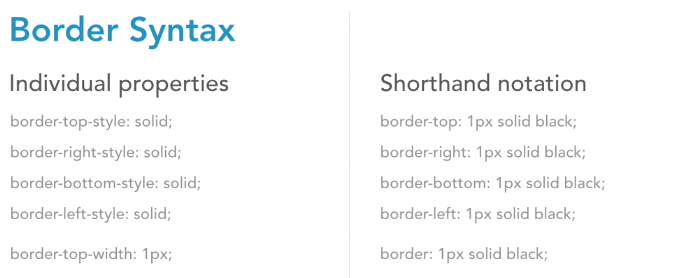




Unlike horizontal margins, vertical margins collapse. That means that if you have two elements stacked on top of one another, only one of the elements margins are applied.

Border

El borde está compuesto por un tamaño, un estilo y un color.



Color

Hay tres formas comunes de definir un color:

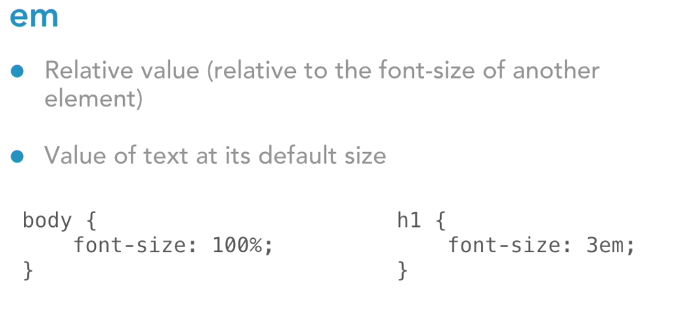
* Por palabra clave
* Por HEXADECIMAL
* Por RGB



También para tener en cuenta hay una propiedad *Opacity: 0 – 1.* Para hacerlo transparente.

Unidades de medidas.

* Px = Valor absoluto, tiene un valor físico.
* Em = Valor relativo. Debido que la mayoria de navegadores tienen por defecto un *font-size* de 16 es posible afirmar que 1 em equivale a 16 px. Pero es escalable, Relativo al *Font-size* **de otro elemento padre**

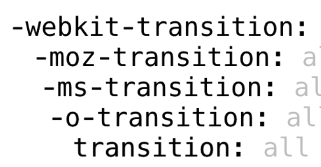


**Body** = **1em (16px)**  
**Hijo** = **1.3em (16px** x **1.3** = **20.8px)**  
**Nieto** = **1.3em (20.8px** x **1.3** = **27.04px)**

* Rem = Se le conocen como *root ems*. Valores relativos. Relativo a elemento raíz. Parecidos a los *em* pero no son escalables.
* % = no se usa mucho en *Font-size*

Vendor Prefixes

Permite a los manufactureros una forma de añadir soporte para nuevos features de CSS antes de que esos features sean totalmente soportados en todos los navegadores.

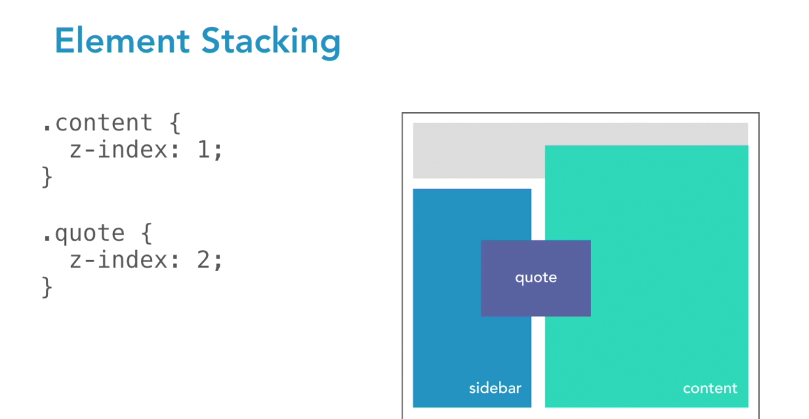


* *-Webkit-: Chrome Safari IOS and Android*
* *-moz-: Mozilla*
* *-ms-: Microsoft IE*
* *-o-: Opera*

*La primera parte representa el manufacturero del navegador*

*Después del (-) se usa la propiedad cuestionada. (En este caso transition)*

## Capítulo 4. CSS Layout



## Capítulo 5. Working with CSS

Un artículo que permite identificar la forma de nombrar las clases:

Class naming methodologies: <https://webdevstudios.com/2017/03/28/evolution-css-class-naming-methodologies/>

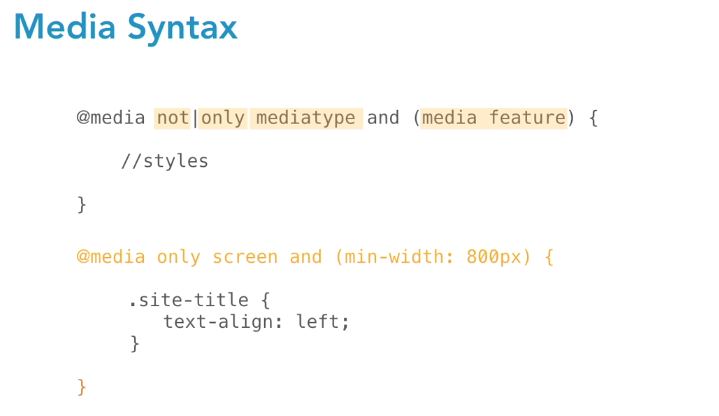
## Capítulo 6. Responsive CSS

Responsive Web Design: Accessible to all types of device.

@Media

Es un reglar de CSS que se puede usar para aplicar parte de una hoja de estilo basado en el resultado de uno o varios ***media queries****.* Existen varios tipos de ***Media Types:***

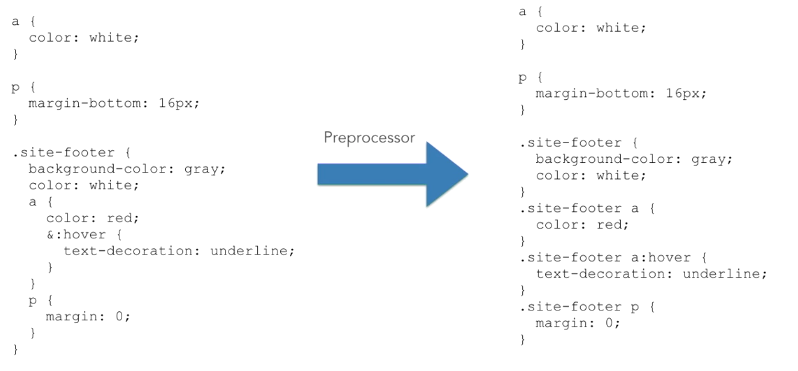
* All: para todo tipo de equipos
* Print: específicamente para material impreso
* Screen: el display de una pantalla
* Speech: para elementos como lectores de pantalla



¡Siempre primero pensar en Mobile Design!

## Capítulo 7. Going further with CSS

* **CSS Grids**
* **CSS Preproccessor:** Un lenguaje de scripting que extiende CSS, ayuda a compilar el CSS que está escrito de otra manera. Los pre-procesadores más comunes son SASS, LESS y Stylus



¿Cuál es el mejor?: <https://www.slant.co/topics/217/~best-css-preprocessors-postprocessors>

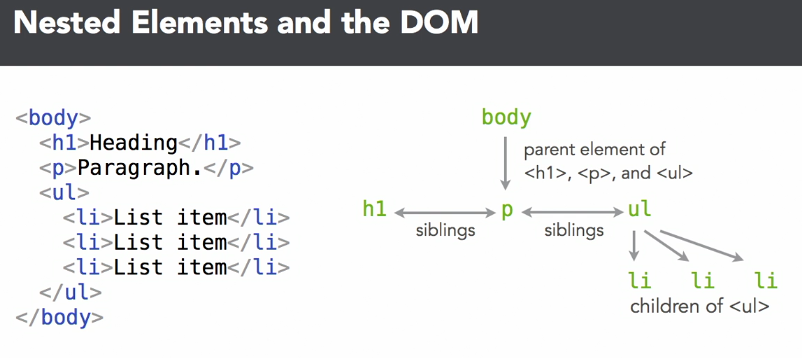
# 2.2 CSS Essential Training 1

## Capítulo 1. Getting Started

En la siguiente imagen se describe las diferencias de las tres principales herramientas del diseño web.

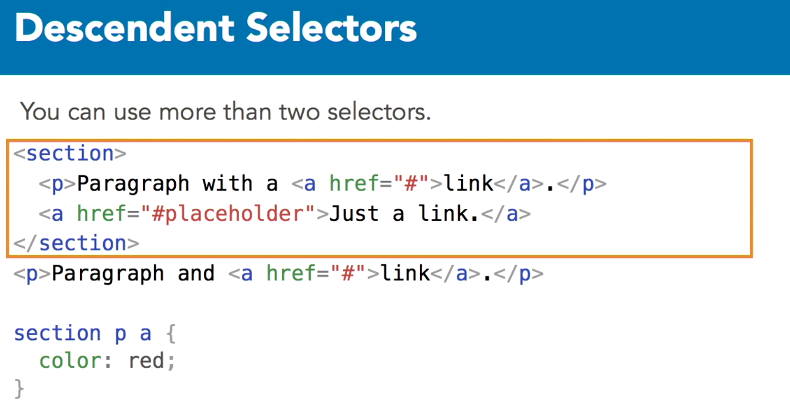


* *CANVAS API*: El elemento <canvas> también se usa en [WebGL](https://developer.mozilla.org/es/docs/Web/WebGL) para dibujar gráficos 3D con aceleración por hardware en páginas web. Estos gráficos se dibujan mediante scripting en JavaScript.
* *DOM (Document Object Model)*: Define la estructura lógica de los documentos y el modo en que se accede y manipula un documento



## Capítulo 2. CSS Core

* **Selectores descendientes:**



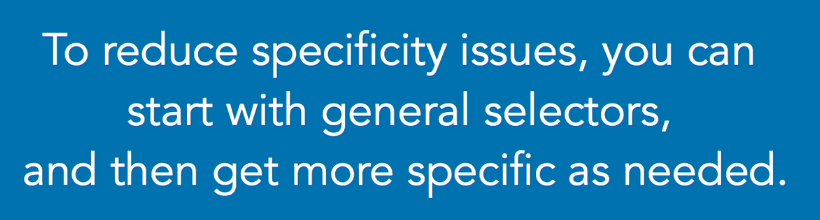
<https://coolors.co/app> = Random Color Palette



Problemas de especifidad:

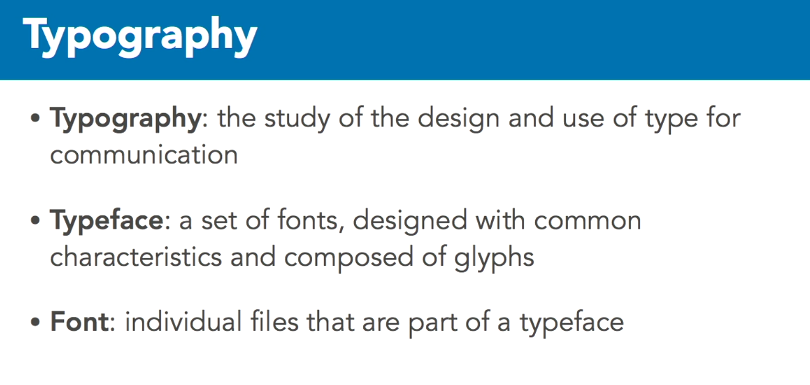
Especificidad son los medios por los que el navegador decide qué valores de una propiedad de CSS son más relevantes para un elemento y, por lo tanto, serán aplicados. La especificidad está basada en las reglas que se aplican y están compuestas de diferentes formas de selectores CSS.

*Revisar calculadora de especifidad:* [*https://specificity.keegan.st/*](https://specificity.keegan.st/)



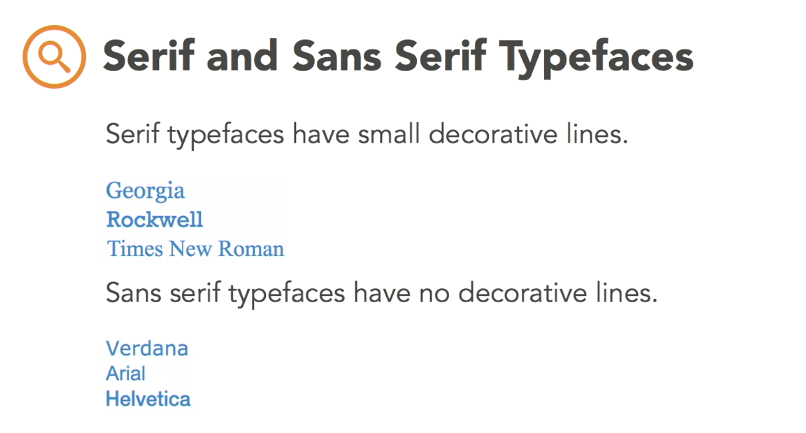
## Capítulo 3. Typography

A continuación, se da la definición de tres conceptos importantes trabajando con fuentes:

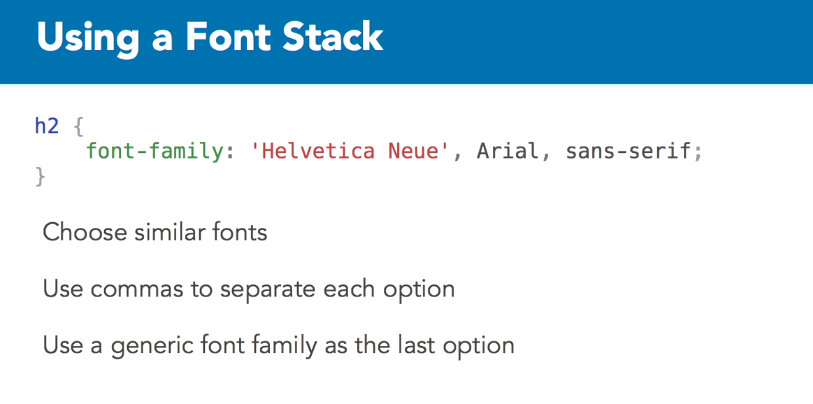


* Tipografia: Estudio del diseño y uso del tipo de comunicación
* Typeface: Un set de fuentes diseñadas con unas caracteristicas comunes, se les conoce como ***font****-****family***.
* Font: Fuente individual parte de un *typeface* o *font-family.*

Diferencia entre Serif y Sans Serif

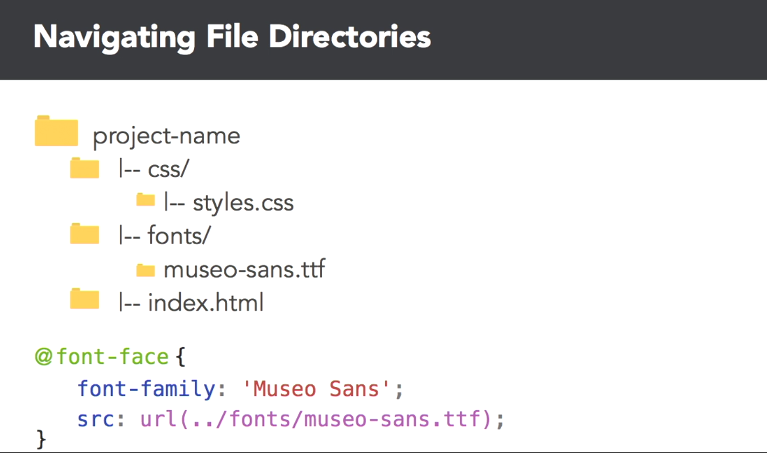


¿Cómo se aplican estos estilos?



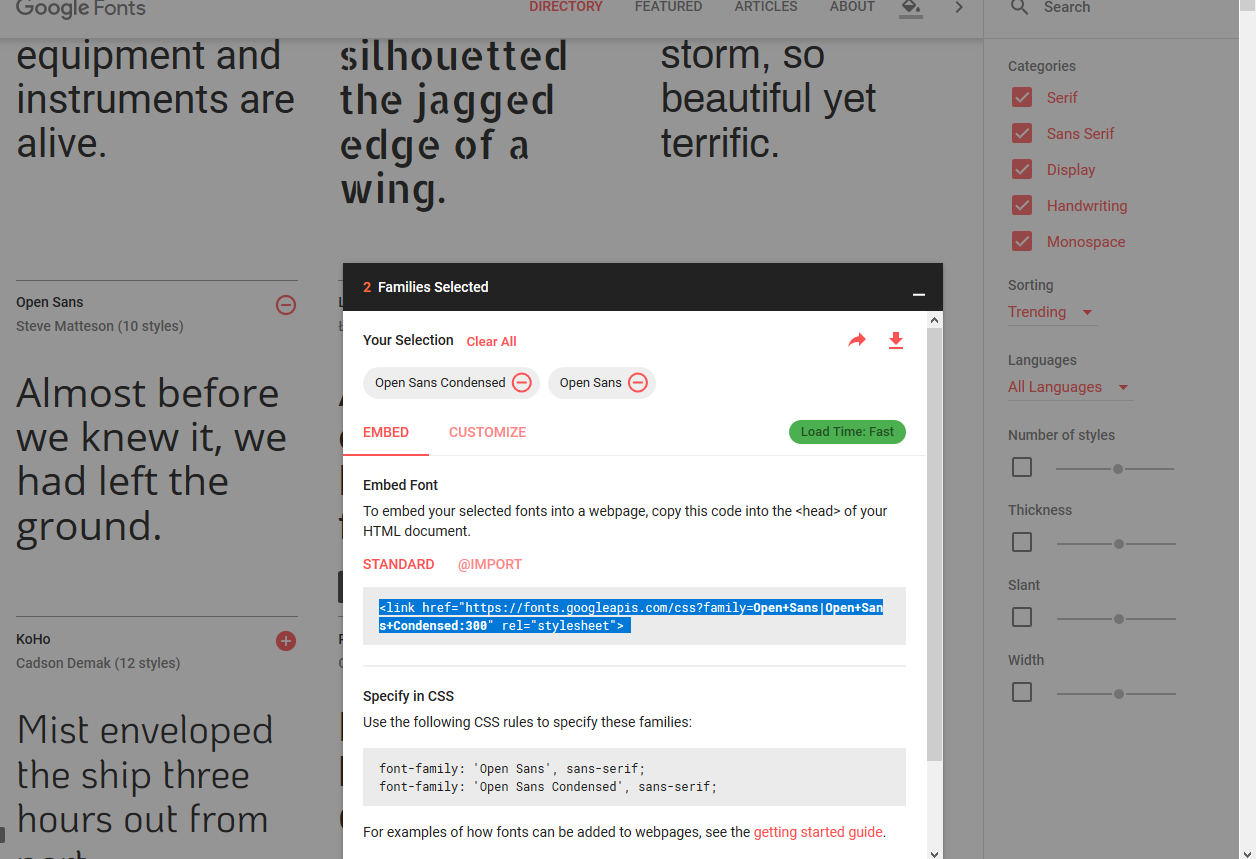
* **Internal Fuentes adicionales**

Se puede también usar un font descargado (INTERNAL). En la imagen muestra como se hace referencia usando la etiqueta *@font-face* e indicando el *font-family* y el *src* donde se encuentra localmente.



* **External fuentes adicionales.**

¡Como Google Fonts! Es un recurso gratuito y haciendo click en el botón de adicionar, genera un código para añadir la etiqueta <link> antes de nuestro estilo, al HTML



Tamaños de Font-Size

**PX= absolutos, unidad física**

**EM = relativa, 1em = default = 16px en ordenador 🡪 Inheritance**

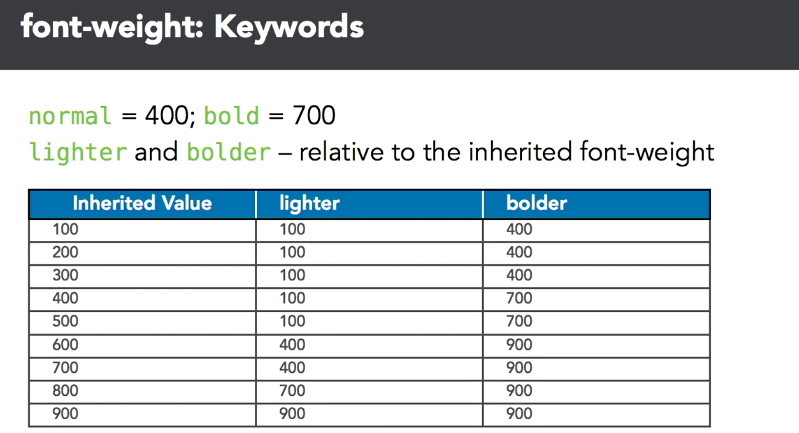
**REM = 1rem = 1em solo hedera a HTML root element**

Font-Weight:

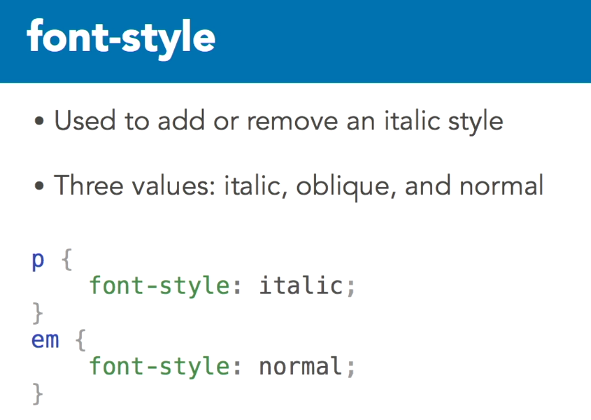
Así se puede definir si tiene negrilla o no.

**Normal = 400**

**Bold = 700**



Font-Style:



Line-height

Es el tamaño del interlineado del texto.

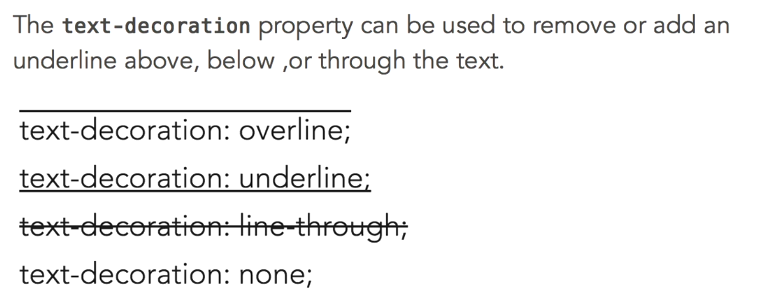


Usando el porcentaje, el interlineado será siempre relativo. (También 1.5 sin unidad tiene el mismo efecto)



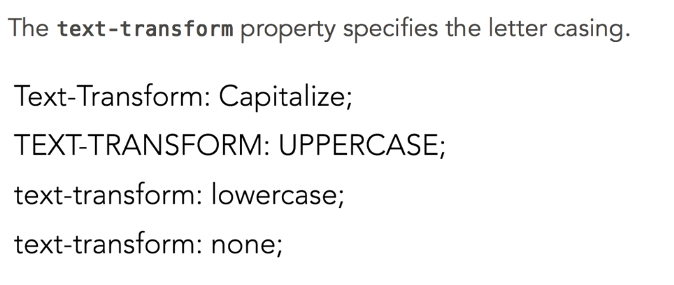
Text-decoration

Se usa para subrayar el texto.



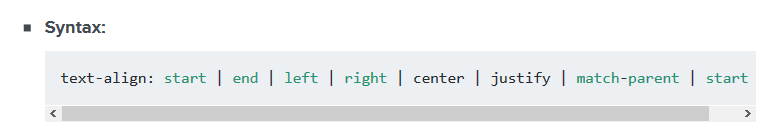
Text Transform

Especifica el letter casing del párrafo.



Text Align

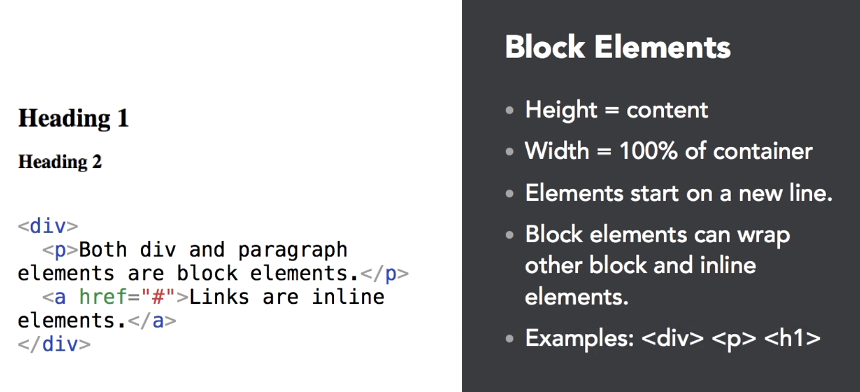
Describe como los contenidos *inline* como los textos o imágenes dentro de un elemento *block* se alinean.



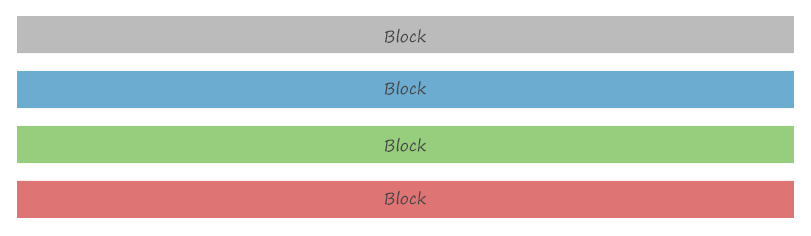
## Capítulo 4. Layouts

Importante entender la diferencia de los tipos de layouts de los elementos:

Block Elements:



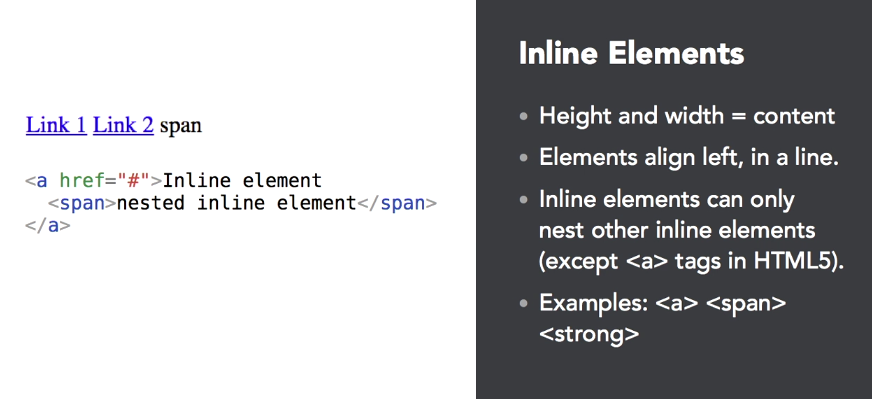
Los elementos a nivel de bloque (block) generalmente son contenedores, y estos ayudan a formar estructuras en el documento web, como son contenedores pueden envasar otros elementos de nivel bloque, estos elementos tienen características extendidas, por ejemplo: márgenes internos y externos, bordes, se le pueden aplicar altura y anchura (Propiedades del box model). Estos son representados en una nueva línea en el documento por lo que no puede tener otros elementos a sus lados, ya que ocupa el ancho de la línea disponible.



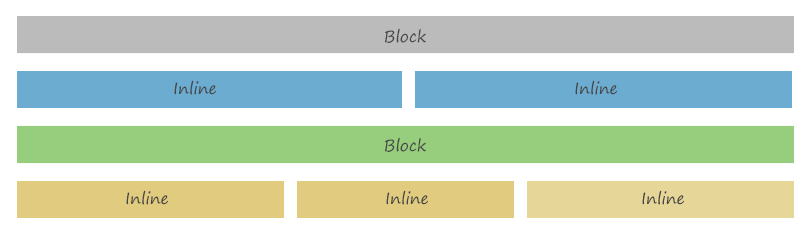
**Elementos a nivel de Bloque:**

article, aside, blockcode, blockquote, body, button, canvas, caption, col, colgroup, dd, div, dl, dt, embed, fieldset, figcaption, figure, footer, form, h1 - h6, header, hr, li, map, object, ol, output, p, pre, progress, section, table, tbody, textarea, td, tfoot, th, thead, tr, ul, video

Inline Elements:



Los elementos en línea (inline) se posicionan en el documento sin afectar su normal flujo, estos no pueden obtener las características que ofrecen las propiedades que dimensionan al elemento y tampoco rompe la línea como si o hace los elementos a nivel de bloque. Por lo general un elemento en línea solo puede contener texto u otros elementos en línea y como dije anteriormente este se renderiza en la línea donde se posicione sin trastornar el flujo en el documento y solo ocupará el tamaño que este se tome.

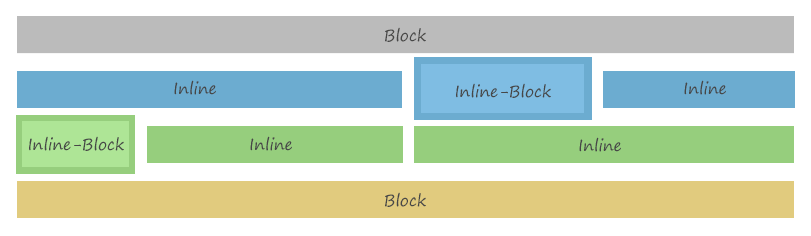


**Elementos HTML a nivel de Línea (inline):**

a, abbr, address, area, b, cite, code, del, details, dfn, datalist, em, font, i, iframe, img, input, ins, kbd, label, legend, link, mark, meter, nav, optgroup, option, q, samp, small, select, source, span, strong, sub, summary, sup, textarea, tt, u, time

Inline-block:

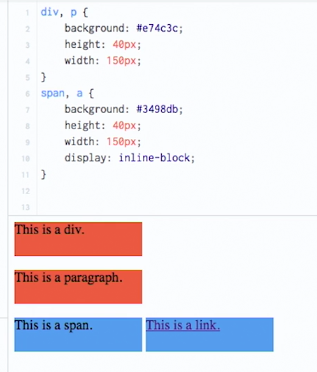
Son los elementos que siguen el flujo de líneas en el documento HTML sin afectarlo, pero que a su vez herede las características de un elemento de bloque.  Elementos como button, del, iframe, ins, map, object, y script pueden ser utilizados tanto como de forma de bloque (recibe las propiedades que afectan el box model) o de forma inline (sigue el flujo de líneas en un documento).



Display:

Los más comunes son:

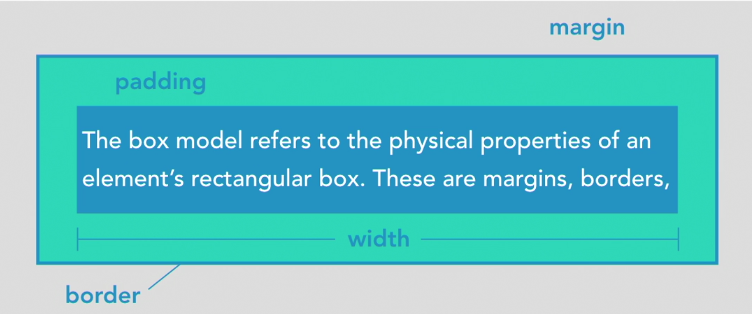
* **Inline**: Hace que se comporte el elemento como un inline
* **Block**: Hace que se comporte el elemento como un block
* **“Inline-Block”:** Toma los valores de Height y Width que un elemento *block* tiene y se organizan de lado como los elementos *inline.* La siguiente imagen lo muestra mejor:



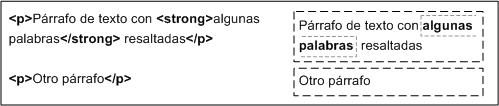
También existe el ***display: none*** que oculta el elemento.

The Box Model – Modelo de cajas

Se refiere a las propiedades físicas de un elemento. Es probablemente la caracteristica más importante de CSS. Hace que todos los elementos de las páginas se representen mediantes cajas rectangulares.

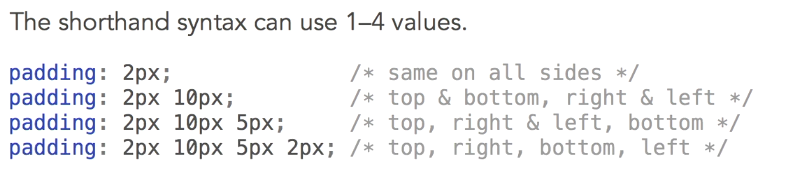


Las cajas se crean automaticamente. Cada vez que se inserta una etiqueta de HTML, se crea una caja rectangular que encierra el contenido de ese elemento, ejemplo:

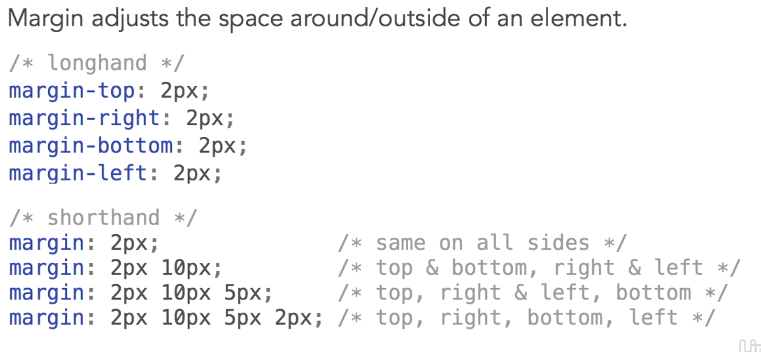


Elementos del Box Model – Modelo de cajas

* **Width and Height:** Son usados para cambiar el tamaño por defecto de elementos *Block* o *Inline.* Especifica el tamaño para el contenido de la caja.
* **Padding:** Ajustar el espacio entre el contenido y el borde. Es un “relleno”

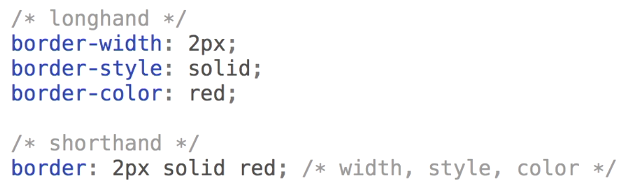


* **Margin:** Ajusta el espacio alrededor/afuera de un elemento, es decir entre la caja existente y el resto adyacentes.



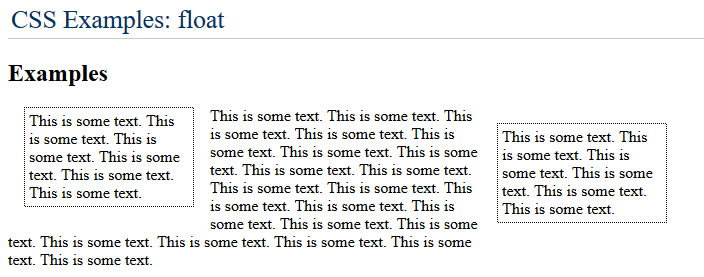
Margin: 0 auto: ¡Logra centrar el contenido!

* **Border:** Es el borde entre el *margin* y el *padding.*

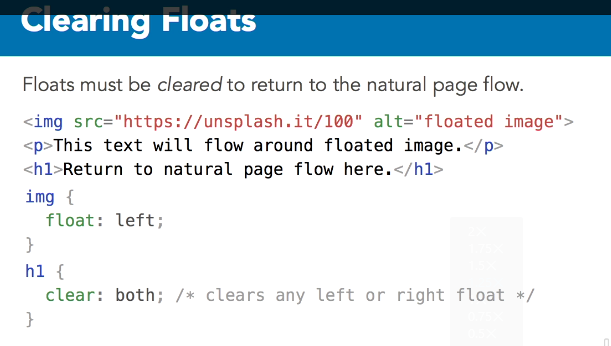


Float and Clear

La propiedad float especifica si un elemento debe salir del flujo normal y aparecer a la izquierda o a la derecha de su contenedor, donde los elementos de texto y los en línea aparecerán a su alrededor.



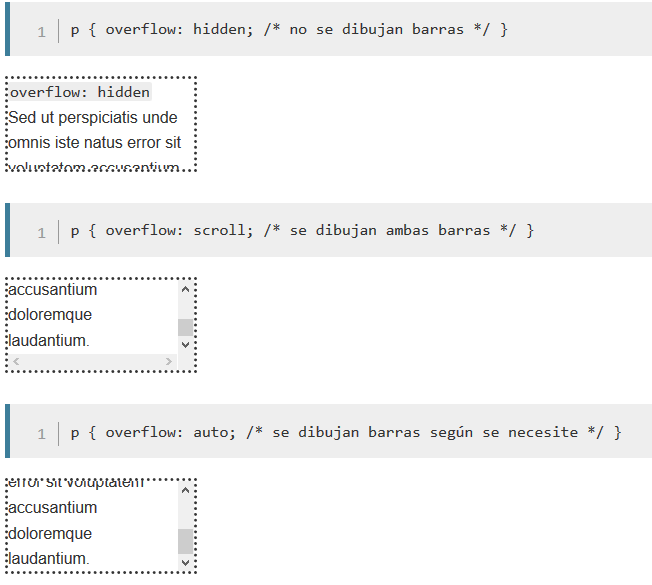
Para volver a Flow natural de la página se debe usar la propiedad de clear.



Overflow

La propiedad CSS overflow especifica si:

* Recortar contenido,
* dibujar barras de desplazamiento o
* mostrar el contenido excedente en un elemento a nivel de bloque.



border-radius:

Vuelve una imagen en un circulo o con esquinas redondas si es una imagen cuadrada. Para obtener el circulo se indica de la siguiente manera:

border-radius: 50%;



# 2.3 CSS Essential Training 2

## Capítulo 1. CSS Selectors

Child combinators

Selecciona únicamente el elemento *child* usando el simbol de mayor (>) entre los selectores *parent* y el *child*

**Ejemplo:**

En base al siguiente HTML:



Si se usa este código todos los *Anchor* **descendientes** de *Sections* tendrán el estilo indicado



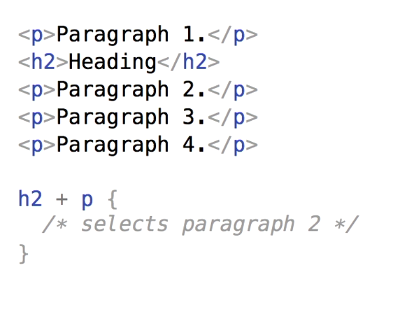
Por otra parte, al usar el estilo de hijo, **solo los elementos hijos directos** tendrán el estilo mencionado:



Sibling combinators

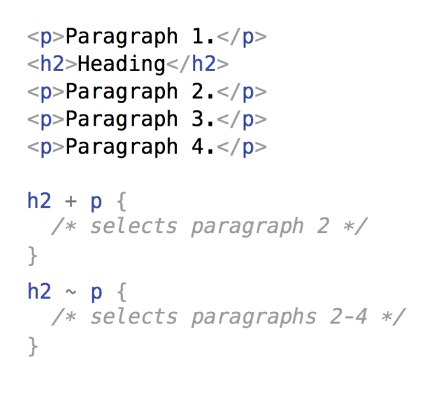
* (+) Hermano Adyacente

En el ejemplo, solo se aplica al párrafo 2.



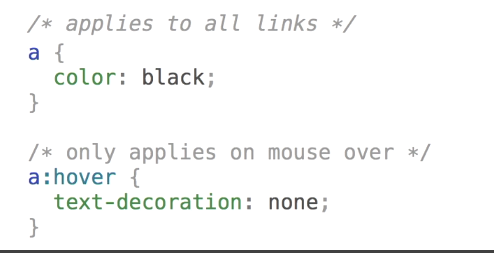
* (~) General

Selecciona cualquier hermano seguido al selector principal



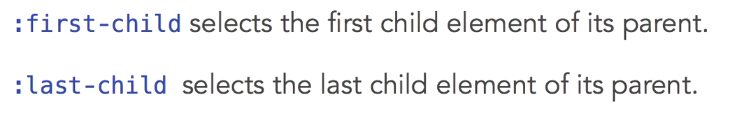
Pseudo Class Selectors

Una pseudo clase es una palabra clave *keyword* que se le añade al selector con el símbolo de dos puntos (:) Se usa para especificar un estado.



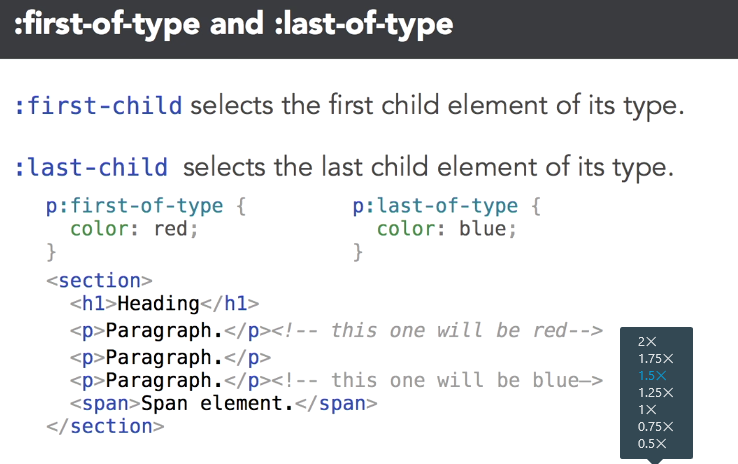
**First and Last child:**

Hay unas pseudo clases también importantes para seleccionar selectores *child.* Para que funcione el tipo de selector debe ser el último *child* de los elementos.



**First and Last of type:**

Sirve para seleccionar el

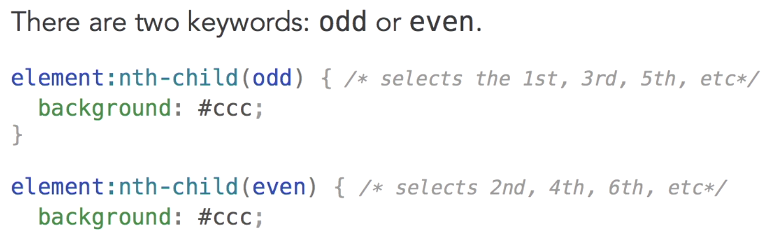


**:nth-child (palabra clave): NTH = SomethingTH term.**



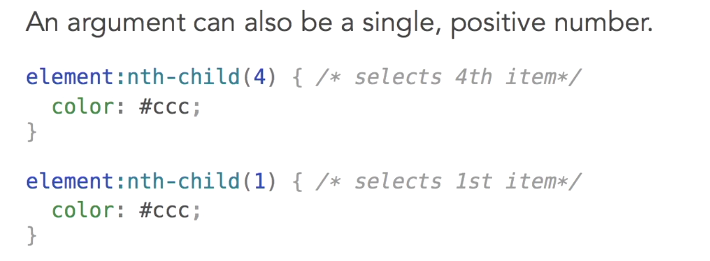
La *Keyword* puede ser:

* **ODD – EVEN**

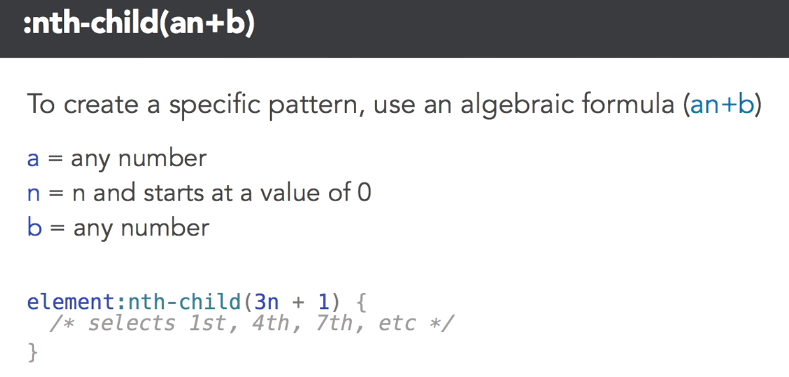


* **NUMBER #**

Seleccionar un *elemento child* específico.



* **UNA FORMULA ALGEBRAICA**



Usando los mismos argumentos, también existen:

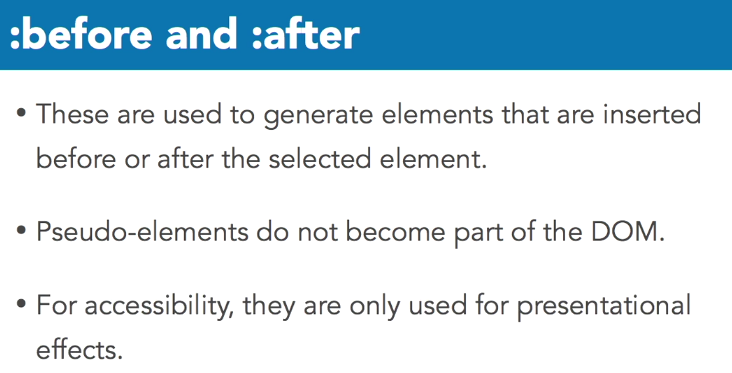
* [:nth-child()](https://developer.mozilla.org/en-US/docs/Web/CSS/%3anth-child)
* [:nth-last-child()](https://developer.mozilla.org/en-US/docs/Web/CSS/%3anth-last-child)
* [:nth-last-of-type()](https://developer.mozilla.org/en-US/docs/Web/CSS/%3anth-last-of-type)
* [:nth-of-type()](https://developer.mozilla.org/en-US/docs/Web/CSS/%3anth-of-type)

Selectores pseudoelementos

Se usan para seleccionar partes de el elemento que no son explícitamente parte del DOM. Al igual que las [pseudo-classes](https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes), los pseudo-elementos se añaden a los selectores, pero en cambio, no describen un estado especial sino que, permiten añadir estilos a una parte concreta del documento. Por ejemplo, el pseudoelemento [::first-line](https://developer.mozilla.org/es/docs/Web/CSS/::first-line) selecciona solo la primera línea del elemento especificado por el selector.



**Hay dos muy importantes:**



**Ejemplo:**



## Capítulo 2. Layouts

Propiedades: Float, Display y Position

*No se usan en un mismo elemento*

* **Position:** Determina en que manera un elemento o item es posicionado en la página o relativo a otro. Por defecto, se encuentran como estáticos, lo que significa que sigue el flujo normal de la página y se ubican en el orden que son definidos en el documento.
* **Display:** Cuando se crea un elemento para mostrar en la página, se representa como una caja. La propiedad CSS **display** especifica el tipo de cuadro de representación utilizado para un elemento.
* **Float:** Para contenido variable y flexible como imágenes rodeadas de texto.

<https://medium.com/@mautayro/understanding-css-position-display-float-87f9727334b2>

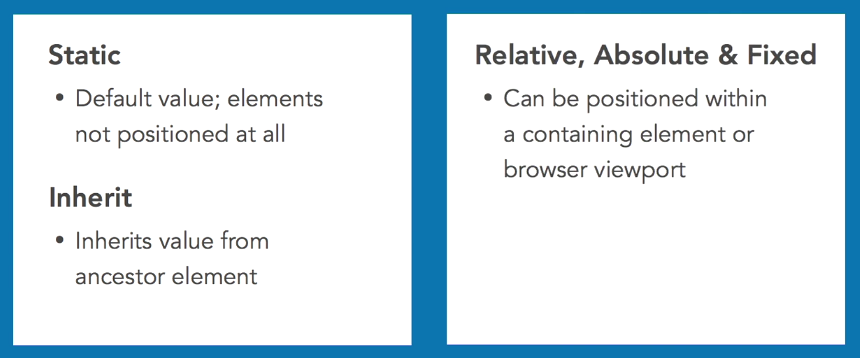
**Display y Float** se tratan a fondo en el [**CSS Essential Training 3 - Capítulo 1.**](#_Capítulo_1._Layout)

Propiedad 🡪 Position

<https://tympanus.net/codrops/css_reference/position/>

Esta propiedad especifica cómo un elemento se posiciona dentro del documento.

**Existen 5 valores para esta propiedad**

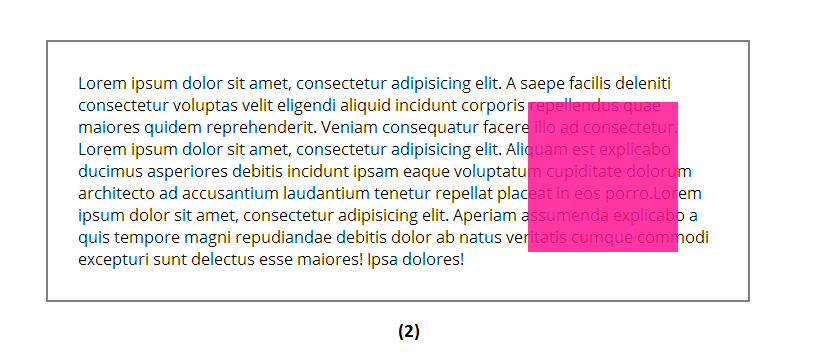


En adición a los cinco valores mencionados, existen dos nuevos añadidos en CSS3: page and center.

* **Static:** Valor por defecto, sigue el flujo normal de la página. Es decir que este tipo **NO** **se identifica como un elemento posicionado.** Las propiedades [top](https://developer.mozilla.org/es/docs/Web/CSS/top), [right](https://developer.mozilla.org/es/docs/Web/CSS/right), [bottom](https://developer.mozilla.org/es/docs/Web/CSS/bottom), [left](https://developer.mozilla.org/es/docs/Web/CSS/left), and [z-index](https://developer.mozilla.org/es/docs/Web/CSS/z-index) no tienen efecto

***Si se cambia el valor por defecto*** *(static), se usan las propiedades de OFFSET (Top, rigth bottom and left) para determinar la ubicación final de los elementos posicionados.*

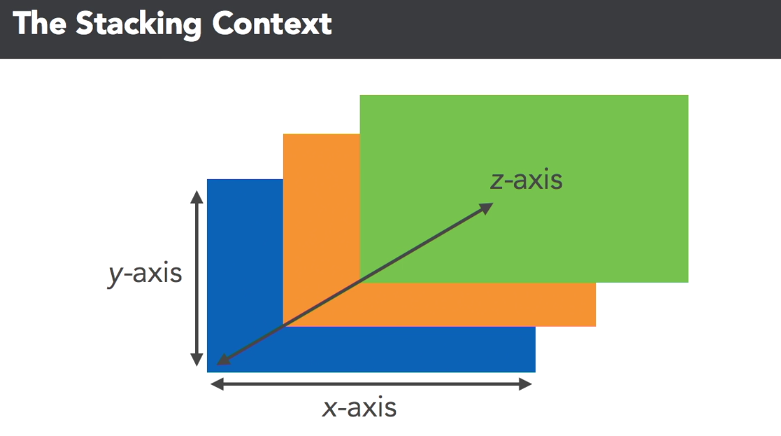
* **Relative:** El elemento es posicionado de acuerdo al flujo normal del documento (mantiene su espacio como si no se hubiera movido), y luego es desplazado con relación a sí mismo, con base en los valores de top, right, bottom, and left. Establece un contexto de posicionamiento a sus descendientes que usen la propiedad **absolute.**
* **Absolute:** El elemento es removido del flujo normal del documento, sin crearse espacio alguno para el elemento en el esquema de la página. Es posicionado relativo a su ancestro **posicionado** (Es decir que tenga una propiedad de posición) más cercano, si lo hay; de lo contrario, se ubica relativo al [bloque contenedor](https://developer.mozilla.org/en-US/docs/Web/CSS/All_About_The_Containing_Block) inicial. Su posición final está determinada por los valores de top, right, bottom, y left. Ancestro en color azul y elemento posicionado en amarillo.



El elemento se posiciona con la propiedad **absolute** en el contexto de posicionamiento de su padre (bordes grises)

* **Fixed:** Se queda aun cuando se hace *scroll* en la página. Su posicionamiento es relativo al **viewport.** Se usa para mantener ciertos elementos en vista todo el tiempo.
* **Sticky:** Un elemento nuevo a tener en cuenta. Comienza actuar como un elemento posicionado ***relative*** y en cierto punto de *scrolling* actua como un ***fixed***.

Stacking context

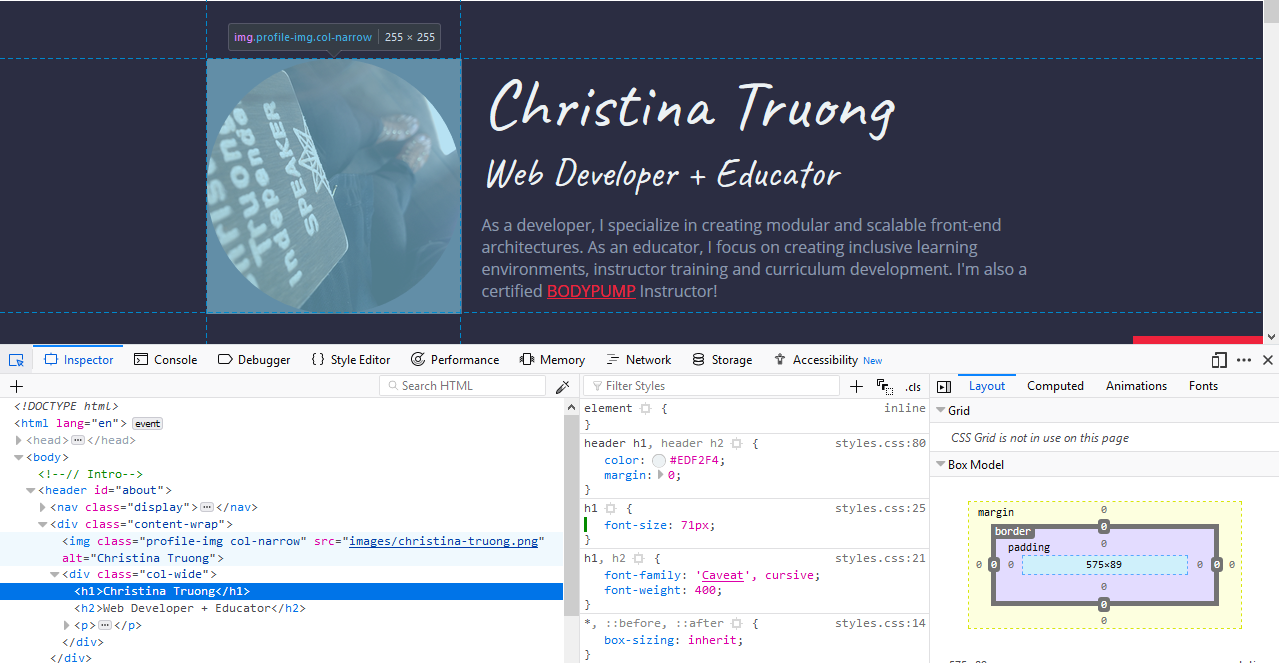


Entre mayor el número del *z-index,* mayor posicionamiento encima de los otros elementos.

## Capítulo 3. Tips and Tools

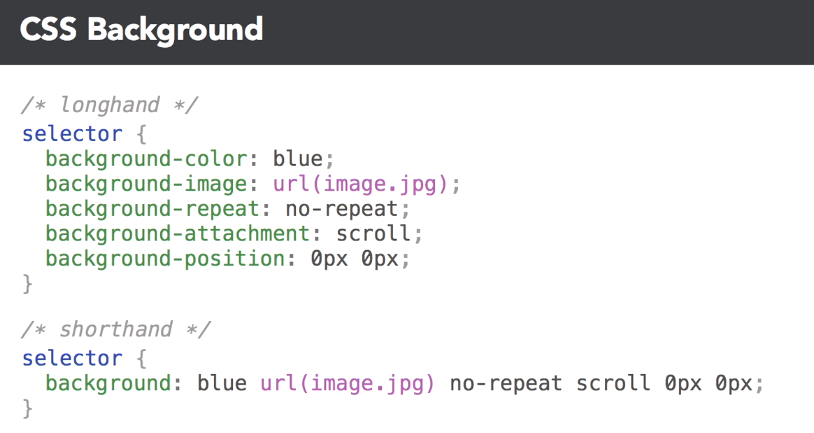
Las herramientas de desarrollador de los navegadores (Que se abren con F12) son de gran utilidad. Entre sus usos más comunes están:

* Usar el elemento de elegir un elemento en la pantalla ubicado de primeras en el NAV de la herramienta (Al lado izquierdo de Inspector)



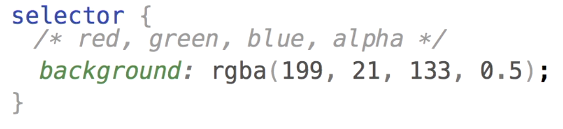
* Hacer directamente en la casilla del CSS para remover temporalmente una propiedad y revisar los cambios que existen.
* Si hay propiedades tachadas significa que están siendo invalidadas (*overriden*) por otra declaración de CSS

Propiedad Background



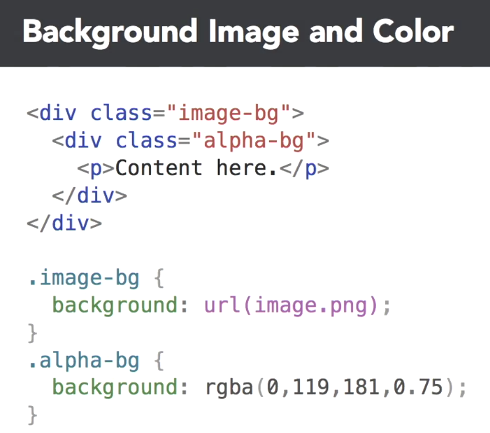
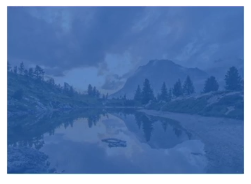
Transparencia Alpha y gradientes

Para mejorar la lectura se puede usar la transparencia Alpha (0-1). Se usa con RGB y el último valor define la opacidad del color.



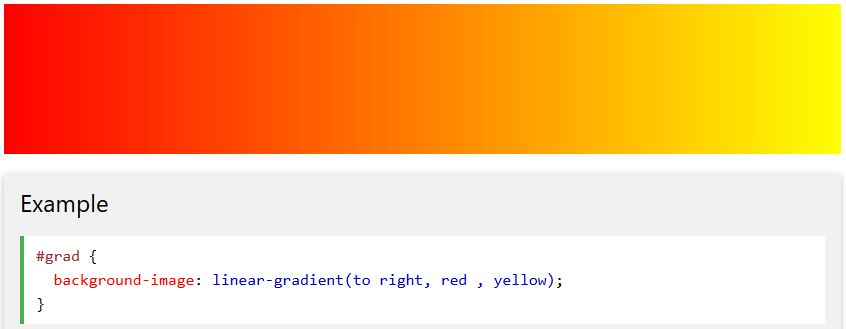
La forma en que se usa esta técnica es la siguiente:

* **Transparencia Alpha**

* **Gradiente CSS**:

Con un gradiente se logra la transición de un color a otro. Al menos, se deben definir dos colores. Se puede definir incluso dirección o ángulo junto al efecto.



## Capítulo 4. Responsive design

Medias Queries

<https://developer.mozilla.org/es/docs/CSS/Media_queries>

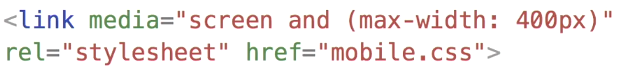
Es la creación de condiciones para aplicar ciertos estilos dependiedo de la consulta realizada.

Los *media queries* consisten de un *media type* y una o más expresiones:

* **Media Types:** Diferentes equipos

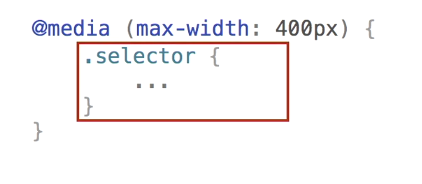
Print – Speech – Screen – All

Estos se pueden añadir desde la etiqueta de <link> en el HTML para cargar ciertos archivos de CSS a la página.

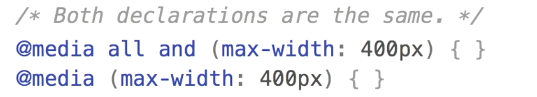


Aquí cargará estilos con una pantalla que tenga un máximo de ancho de 400px.

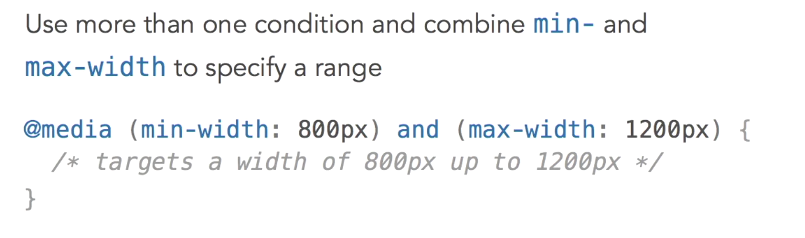
Asimismo, pueden ser añadidos dentro del CSS, como se muestra a continuación:



Si **NO se especifica una media *type***por default se selecciona a *all*



También se pueden usar varias condiciones siempre y cuando estén separadas por la palabra ***and***



Etiqueta <Meta name=viewport>

<meta name="viewport" content=" user-scalable=no, width=device-width, initial-scale=1.0">

Esta etiqueta da instrucciones a los exploradores de internet de dispositivos móviles para ajustar el viewport y mostrar correctamente la página web:

La etiqueta **Content** agrupa varias propiedades como:

* width=device-width: Asegura que la pagina se extienda el ancho en el dispositivo
* initial-scale=1.0: Es el valor del zoom. Este valor asegura que quede no haya ningun zoom.
* user-scalable=no: Se consigue que el usuario no pueda hacer zoom en la página.

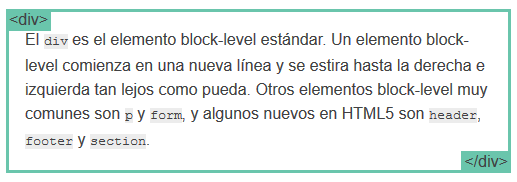
# 2.4 CSS Essential Training 3

## Capítulo 1. Layout

Propiedad Display

Cuando se crea un elemento para mostrar en la página, se muestra como una caja y cada elemento tiene un valor por defecto. Con esta propiedad de **display** se puede especificar el tipo de caja.

* **Block:** Se define como una caja que se posiciona sobre otras cajas. Comienzan en una nueva linea y toman el ancho entero. Pueden conteneder otros elementos tanto como **block e inline.**



Algunos elementos que son *block-level* por defecto son:

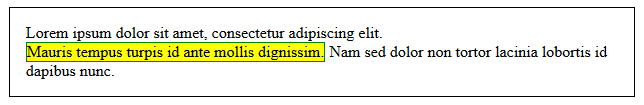
<div>, <p>, <h1>-<h6>, <ul>, <li>, & <canvas>.

* **Inline:** Es lo opuesto a un elemento tipo bloque. Pueden empezar en cualquier parte de una linea existente. Las propiedades de Alto y Ancho no tienen ningun efecto. Un elemento inline puede contener algo de texto dentro de un párrafo <span> como esto </span> sin interrumpir el flujo del párrafo

Algunos elementos que son *block-level* por defecto son:

<span>, <input>, <button>, &<img>.

* **Inline-Block:** Es una mezcla entre los dos primeros. El elemento aparecerá de forma *inline* pero el alto y el ancho pueden ser definidos



* **None:** Es un valor común que elementos especiales como <script> lo usan por defecto. Es comúnment0e usado en JavaScript para ocultar elementos sin eliminarlos o recrearlos. En el ejemplo la caja va a la segunda linea manteniendo la forma de caja porque no hay suficiente espacio en la primer linea.

Esto es diferente de visibility. Usar display: none no dejará espacio donde el elemento se encontraba, pero visibility: hidden; dejará un espacio vacío.

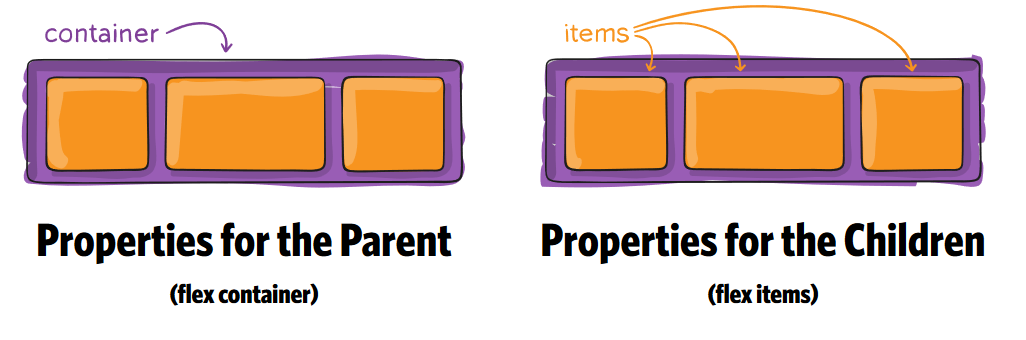
Display: Flex – Conocido como Flexbox

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

Hay una serie de ejemplos creados por mi en la siguiente ruta:

2. CSS\2.3 CSS Essential 3\Exercise Files\Chapter 2\Mis ejemplos

Para usar *Flexbox (Flexible box layout o Caja Flexible)*, lo primero es hacer el valor de *display: flex.* Es una forma de ubicar elementos HTLM de manera automática y más personalizable. Le permite al *container* la habilidad de alterar el ancho y alto de sus elementos, ordenarlos y ubicarlos de manera correcta.



Esta técnica permite colocar los elementos de una página para que se comporten de forma predecible cuando el diseño de la página debe acomodarse a diferentes tamaños de pantalla y distintos dispositivos. **Este modelo según el MDN de Firefox es apropiado para diseños de pequeña escala**, mientras que el modelo de ***grid*** es adecuado para diseños de gran escala.

**Las siguientes son propiedades del CONTENEDOR:**

* **Display: Flex**: Esta propiedad es la primera en definirse. También puede ser *inline-flex.* Esto ayuda identificar al que será el contenedor.

.flex-container{

display: flex; /\* or inline-flex \*/

}

* **Flex-flow:** Esta propiedad es la combinación de las propiedades *flex-direction* (define la dirección de la caja flex: filas o columnas) y de *flex-wrap* (por defecto los elementos se ajuntan en una sola linea, si se usa *wrap* se agruparán en varias lineas). **El valor por defecto es**: flex-flow: *row nowrap*

.flex-container{

display: flex; /\* or inline-flex \*/

flex-flow: row wrap;

}

* **Justify-content:** Define la alineación de los items por el eje principal u horizontalmente hablando de filas. Ayuda a distribuir espacio extra.
* **Align-items:** Define el comportamiento de los items en el eje vertical. Es como un *justify-content*, pero verticalmente
* **Align-content:** Define la alineación de cada una de las filas cuando hay espacio extra en el eje vertical. Esta imagen puede representar mejor su comportamiento:



**Las siguientes son propiedades de los ITEMS FLEX:**

<https://medium.freecodecamp.org/even-more-about-how-flexbox-works-explained-in-big-colorful-animated-gifs-a5a74812b053>

* **Order:** Por defecto los items se muestran en el orden de su origen; sin embargo, se puede controlar el orden definiendo un número entero a cada item.

.item {

order: 1; /\* default is 0 \*/

}

* **Flex:** Esta propiedad contiene las tres siguientes propiedades en el mismo orden:
  + **Flex-grow:** La más importante de las tres. Define la habilidad de un *flex item* de crecer si es necesario. Es un valor relativo entre los items.

El valor 0 hace que los items no puedan crecer, es el valor por defecto.



El valor 1 hace que los items tomen el ancho total y se dividan el espacio de forma equitativa. No obstante, si todas las cajas tienen el mismo valor ya sea 1 o 999, este será el mismo resultado.



Para empezar a ver la diferencia tendriamos que pensar en fraciones. En el siguiente ejemplo la tercera caja tendra un valor de *flex: 3* y el resto *flex: 1*. Entonces, la división del espacio sería la siguiente:

**(1 + 1 + 3 + 1 + 1 + 1) = 8.**

**Caja 3: 3/8 del ancho**

**Resto de cajas: 1/8 del ancho**



* + **Flex-shrink:** Es la propiedad opuesta a *flex-grow*, determinando cuanto un item puede reducirse. Solo se aplica si los elementos deben reducirse para caber en su container; por ejemplo, cuando el contenedor es muy pequeño.

El valor por defecto es 1, lo cual indica que todos los items se reduciran por igual.

Si hay un valor de un item que es igual a 0, indica que este se opone a reducirse.

* + **Flex-basis:** Especifica la base flexible la cual es el tamaño inicial de un item flexible. Determina en otras palabras el tamaño de una caja de contenidos. Su valor inicial es *auto* que significa que toma los valores por defecto asignados y en caso de no tener alguno, toma el alto del contenido.

En el momento de usar la versión corta de estas propiedades; es decir, **FLEX**, lo siguiente aplica:

(flex-shrink y flex-basis) Son opcionales

El valor por defecto es: 0 1 auto.

Display:Grid

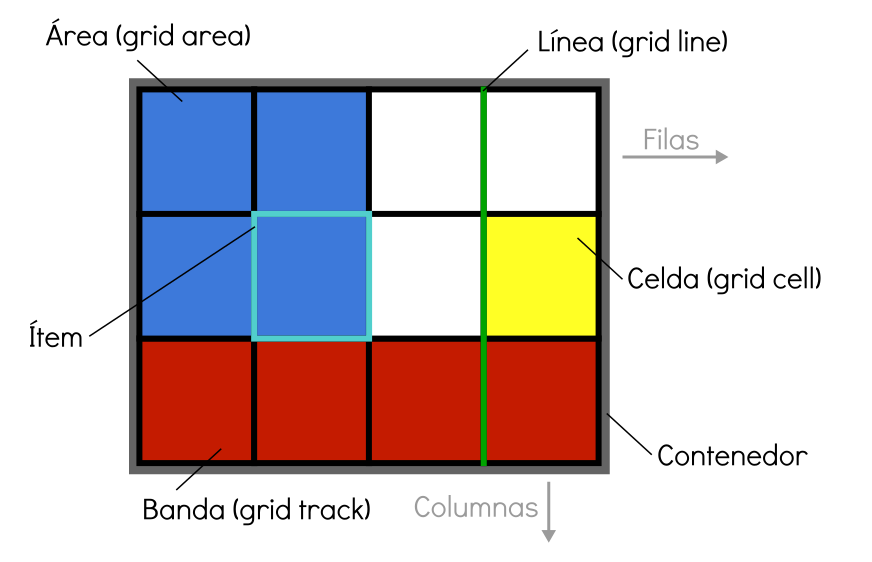
**Para mayor detalle, ver apartado del** [**CURSO CSS GRID**](#_2.5_CSS_Grid)**.**

<https://lenguajecss.com/p/css/propiedades/grid-css>

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

Nace de la necesidad de crear un *layout* como una cuadrícula. Se diferencia con *flexbox* porque esta última está orientada solo a una dimensión. Esta es una alternativa para crear cuadriculas sencillas y más potentes.

La siguiente es la representación gráfica de los conceptos principales de GridCSS:

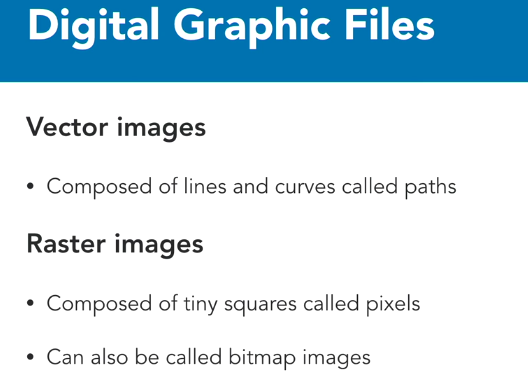


* **Contenedor**: Existe un elemento padre que es el contenedor que definirá la cuadrícula o rejilla.
* **Ítem**: Cada uno de los hijos que contiene la cuadrícula (*elemento contenedor*).
* **Celda (grid cell)**: Cada uno de los cuadritos (*unidad mínima*) de la cuadrícula.
* **Area (grid area)**: Región o conjunto de celdas de la cuadrícula.
* **Banda (grid track)**: Banda horizontal o vertical de celdas de la cuadrícula.
* **Línea (grid line)**: Separador horizontal o vertical de las celdas de la cuadrícula.

FR es una unidad relativa usadas en Grid. Se le conocen como Fraction.

## Capitulo 2. Retina Display and Images

Dos formas de imágenes: **Vectores y ráster**

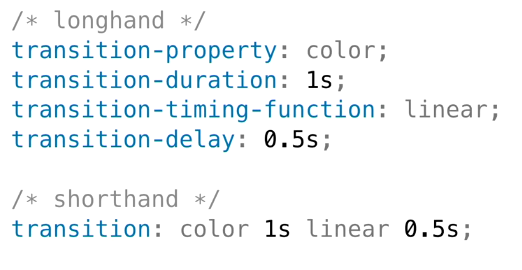


* JPEG: Ráster (cuadros conocidos como pixeles)
* PNG: Ráster (cuadros conocidos como pixeles)
* SVG: Vectores escalables.

## Capítulo 3. Animaciones

Transition

Crea transición de un estado a otro. Se contrala el tiempo y el efecto para hacer el cambio. A continuación, la sintaxis:



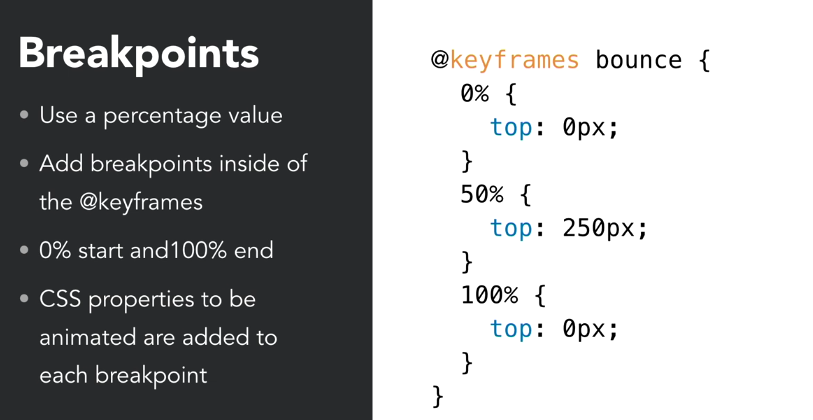
* **Property**: propiedad que se cambiará (No todas son transicionables)
* **Duration**: Duración del cambio
* **Timing**-**function**: Añadir una curva de aceleración cambiando la velocidad de cambio en distinto tiempo, es decir puede empezar rapido y termina despacio.
* **Delay**: Retraso del inicio de la animación.

Keyframes

A diferencia de la propiedad de *Transition,* se puede añadir distintas animaciones más complejas y con distintos puntos de referencia (no solo inicio y fin) usando la propiedad *@keyframes*

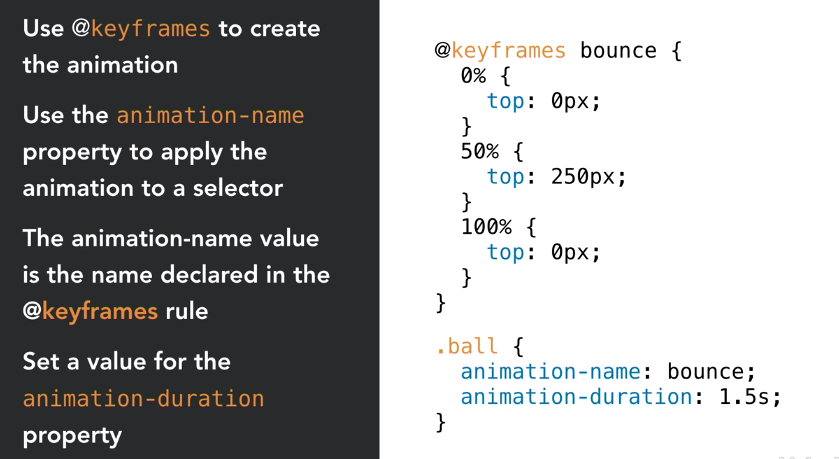


Un breakpoint se define de la siguiente manera:



Para tener en cuenta, una animación solo se puede aplicar si se usa la misma propiedad.

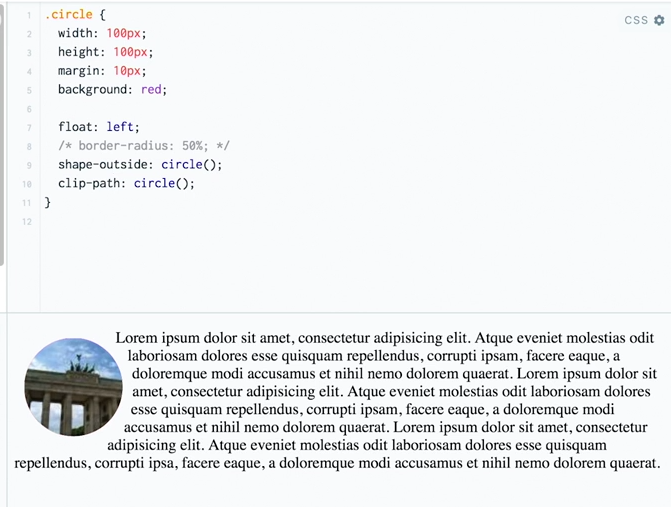
Para usarlo:



Tambien se puede añadir un *delay y un timing function*

CSS Shapes:

Se puede usar para que el texto luzca como en la imagen, y no se considere la imagen como un cuadrado



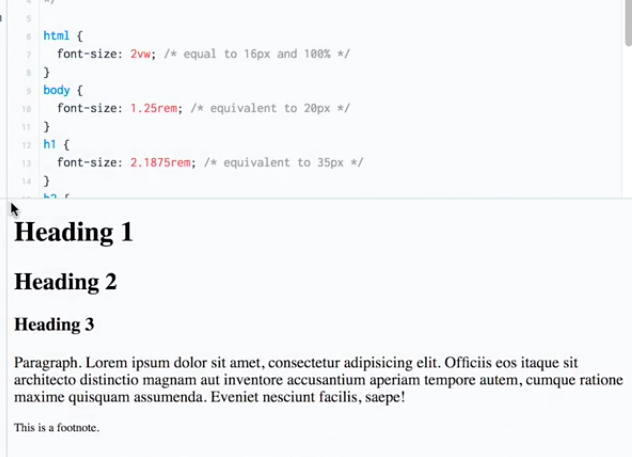
## Capítulo 4. Tools and Tips

Responsive Typography

REM es una unidad relativa al tamaño de fuente del elemento *root* que sería el *html.* Se puede usar para definir el tamaño en toda la página de una forma relativa y usando *media queries* se puede redefinir en caso de tener equipos más pequeños:



También existen unidades relativas conocidas como *Viewport units*



¡Selectores de atributos!

También se puede seleccionar un atributo en sí para referenciarlo en el CSS. Generalmente se puede usar en los *inputs*



Atributo \*= “String”

Una forma de verificar que el string se encuentre en el valor sin importar si se encuentra al principio o final. Es una referencia en CSS de selectores más avanzados.



# 2.5 CSS Grid

<https://lenguajecss.com/p/css/propiedades/grid-css>

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

**Curso de CSS Grid encontrado en:** <https://www.youtube.com/playlist?list=PLM-Y_YQmMEqBxmylkI5WJn9ouUxWlJNOW>

Nace de la necesidad de crear un *layout* como una cuadrícula. Se diferencia con *flexbox* porque esta última está orientada solo a una dimensión. Esta es una alternativa para crear cuadriculas sencillas y más potentes.

## Lección 01 – Práctico Introducción

En la siguiente imagen se muestra los elementos que se quieren alinear dentro de una página. Se desea que el enfoque móvil sea de una fila cada elemento y el enfoque escritorio sea un grid de *3X3* donde tanto como el header y el footer ocupen completamente una fila, y los tres restantes de dividan la fila sobrante. El código HTML para esta lección es el siguiente:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Curso de CSS Grid</title>

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

</head>

<body>

<main>

<header>Header</header>

<aside>Izquierda</aside>

<article>Contenido</article>

<aside>Derecha</aside>

<footer>Footer</footer>

</main>

</body>

</html>

Igual que Flexbox, se define primero un container que se hará usando la clase *display.*

/\* >>[1]<< Definimos el GRID Container

\* El enfoque será first-mobile

\*/

main{

*display*: grid;

*grid-gap*: 10px;}

Una vez hecho esto, se debe asginar a cada uno de los elementos que deseamos manipular dentro del GRID, un nombre de área, esto se hace de la siguiente manera:

/\* >>[2]<< Relacionamos un área a cada elemento. \*/

header{

*grid-area*: header;

}

aside:first-of-type{

*grid-area*: izquierda;

}

article{

*grid-area*: contenido;

}

aside:last-of-type{

*grid-area*: derecha;

}

footer{

*grid-area*: footer;}

Hecho esto volvemos al elemento **main** y le asignamos las áreas del grid. Como es un enfoque *first-mobile* los elementos iran uno encima del otro:

main{

*display*: grid;

*grid-gap*: 10px;

/\* >>[3]<< Definimos la columna en enfoque móvil\*/

*grid-template-areas*:

"header"

"izquierda"

"contenido"

"derecha"

"footer";

}

E inmediatamente, con un media querie podemos asignar el enfoque *escritorio*. El cual será **3x3.** Como se observa en la siguiente imagen, los elementos *header y footer* ocupan toda una fila. Para representar un espacio vacio, solo basta colocar un punto como se muestra en la parte que está comentada.

/\* >>[4]<< Definimos el enfoque para escritorio. Para escritorio sera tres columnas.\*/

@media (*min-width*: 500px){

main{

/\*El grid sera un 3x3\*/

*grid-template-areas*:

"header header header"

"izquierda contenido derecha"

"footer footer footer"

;

/\*Si se coloca un punto, este representará un espacio vacio en el grid. \*/

grid-template-areas:

"header header header"

"izquierda contenido derecha"

"footer footer ."

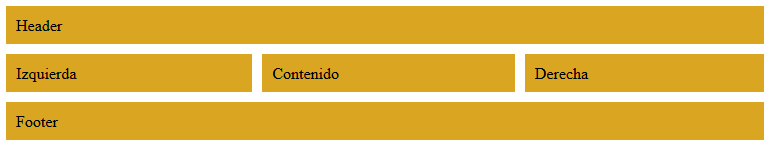
;

}

}

Los resultados obtenidos son:

* Cuando está en el enfoque de escritorio, es decir cuando el ancho sea de mínimo 500 px.



* Cuando es el enfoque móvil:



* Si el template-areas se coloca un punto, este representa un espacio vacio. Asi se vería en el enfoque de escritorio:



Esta lección es solo una pequeña introducción de CSS Grid. Hay muchas más propiedades para el contenedor y los elementos que se mostrarán a lo largo del curso.

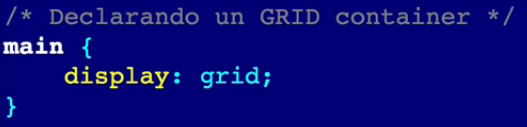
## Lección 02 – Terminologia y primeros pasos

**¡CSS Grid NO sustituye a FlexBox, lo complementa!**

Conceptos esenciales

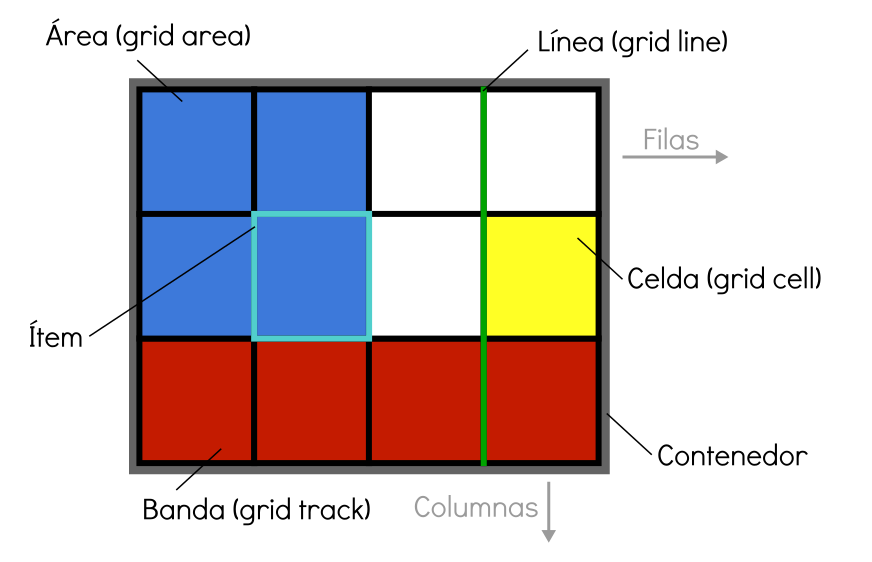
* **Grid Container & Grid Item:**

El grid container es el padre que agrupará a todos los grid ítems dentro de su grid. Esta relación se le conoce como **grid context**.



* **Caja de Grid**

La siguiente es la representación gráfica de los conceptos principales de GridCSS:



* **Contenedor**: Existe un elemento padre que es el contenedor que definirá la cuadrícula o rejilla.
* **Ítem**: Cada uno de los hijos que contiene la cuadrícula (*elemento contenedor*).
* **Celda (grid cell)**: Cada uno de los cuadritos (*unidad mínima*) de la cuadrícula.
* **Area (grid area)**: Región o conjunto de celdas de la cuadrícula, entre 4 grid lines. Puede contener N número de celdas.
* **Banda (grid track)**: Banda horizontal o vertical de celdas de la cuadrícula. Puede ser una fila o una columna, entre dos líneas consecutivas.
* **Línea (grid line)**: Separador horizontal o vertical de las celdas de la cuadrícula. Se puede hacer referencia a una *grid line* por número o por nombre. En la imagen hay 4 lineas horizontales y 5 verticales.

Ejemplo práctico Lección 2

Tenemos en el HTML 10 divs, que servirán para representar gráficamente los ítems del grid. Vamos a trabajar con 10 cuadros.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

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

<title>Curso de CSS Grid - Lección 02</title>

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

</head>

<body>

<main>

<div>1</div>

<div>2</div>

<div>3</div>

<div>4</div>

<div>5</div>

<div>6</div>

<div>7</div>

<div>8</div>

<div>9</div>

<div>10</div>

</main>

</body>

</html>

Estas cajas serán elementos flex que a su vez serán ítems o elementos child del container grid. Esto para mostrar que **Grid y Flex son completamente compatibles** y el primero complementa al último.

div{

*display*: flex;

*justify-content*: center;

*align-items*: center;

*background-color*: coral;

}

Es posible, asignar de otra manera el grid template del container sin definir áreas. Esto se logra definiendo los tamaños de las filas y las columnas con las siguientes propiedades:

* *Grid-template-columns*
* *Grid-template-rows*

main{

*height*: 100vh; /\*Representa el 100% del viewport height\*/

*display*: grid;

*grid-gap*: 1em;

/\* grid-template-columns: 100px 100px 100px 100px;

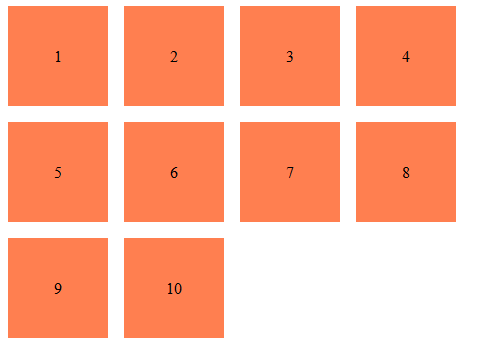
grid-template-rows: 100px 100px 100px; \*/

*grid-template-columns*: repeat(4,100px);

*grid-template-rows*: repeat(3,100px);

}

Como lo muestra el código, se puede usar la función repeat que equivale a los valores con comentarios. En la siguiente imagen, se muestra el resultado obtenido hasta ahora.



Asimismo, usando una pseudoclase, podemos hacer referencia a dos elementos para definir el número de columnas o filas que ocupan.

div:nth-child(10){

*background-color*: greenyellow;

/\*

grid-column-start: 3; Empieza desde la GRID-LINE #3

grid-column-end: 5; Termina en la GRID-LINE #5

\*/

*grid-column*: 3/5; /\*Se puede usar -1 para señalar hasta la última\*/

}

div:nth-child(6){

*background-color*: greenyellow;

*grid-column*: 2/3;

*grid-row*: 2/4;

}

* En la primera parte, a través de la propiedad ***grid-column*** le indicamos desde qué *GRID LINE* (vertical o de columna) empieza a ocupar espacio y hasta qué *GRID LINE* termina (Se puede usar -1 para indicar que llegue hasta la última línea). La forma larga usada está comentada y se usa dos propiedades.
* En la segunda parte, usando esta vez la propiedad ***grid-row*** podemos indicar desde que *GRID LINE* (horizontal o de fila) empieza a ocupar espacio en el grid y hasta que *GRID LINE* termina. Funciona igual que la propiedad anterior.

Los resultados obtenidos, junto a las líneas resaltadas, se muestran en la siguiente imagen:



## Lección 03 – Separación entre los elementos Grid

Un punto de confusión en CSS Grid es que se suele pensar que unidad de medidad *fraction* es igual a *auto*. Por ejemplo, ambos emiten el mismo resultado en este ejemplo:

main{

height: 100vh;

display: grid;

grid-gap: 10px;

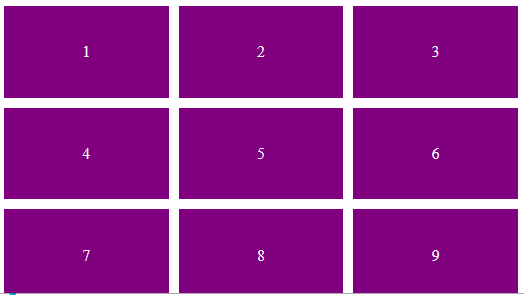
grid-template-columns: 1fr 1fr 1fr;

*/\*\**

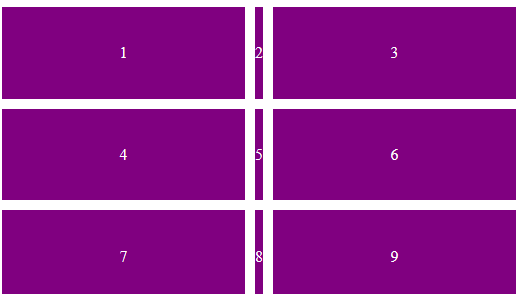
*\* grid-template-columns: auto auto auto;*

*\*/*

}



Sin embargo, si cambiamos una de las fracciones por auto, nos damos cuenta que las fracciones tienen prioridad/precedencia y el auto en realidad tomara el espacio restante después de las divisiones de las fracciones.

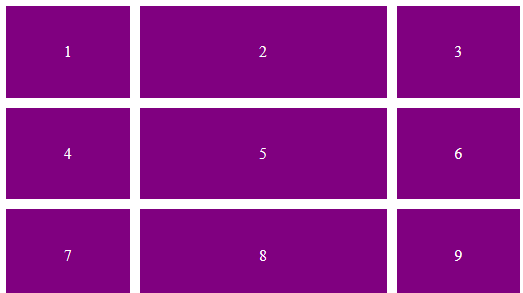


Es por eso que al usar *fraction* se divide el espacio disponible en el número de fracciones indicadas.

grid-template-columns: 1fr 2fr 1fr;

En este caso, se reparte en 4 pedazos el espacio. (1+2+1 = 4) y cada parte le corresponde:

**¼ 2/4 ¼**



Usando función repeat ()

Usando una sintaxis más corta con la función *repeat* obtenemos el grid de 3x3 sin tener que repetir los valores tres veces:

main{

height: 100vh;

display: grid;

grid-gap: 10px;

grid-template-columns: repeat(3, 1fr);

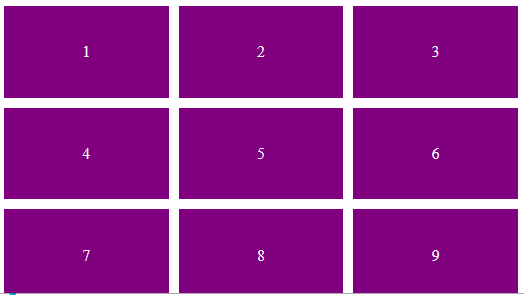
grid-template-rows: repeat(3, 1fr);

*/\*\**

*\* grid-template-columns: auto auto auto;*

*\*/*

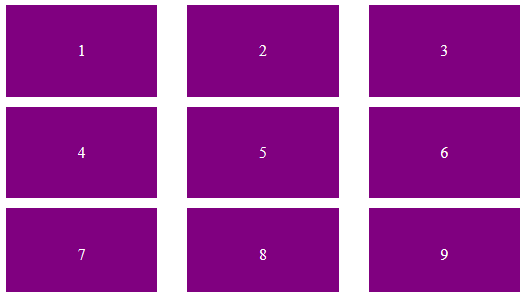
}



Grid-gap

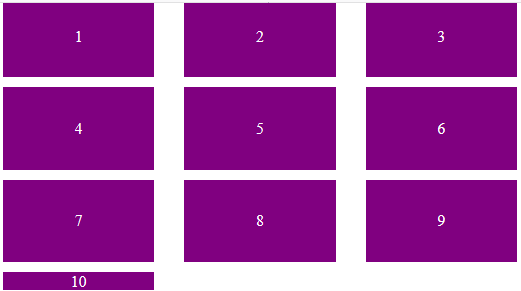
Asimismo, la propiedad *grid-gap* se compone de dos propiedades que defeniran el *gap* entre filas y columnas respectivamente:

grid-gap: 10px 30px;



Implicit Grid

Cuando hay elementos que no están definidos dentro del Grid (Nuestro grid estuvo definido de 3x3 en la parte de las filas) obtendríamos un resultado similar:



Usando una propiedad de implicit grid, podemos asignar el alto de las filas a automatico y asi, sin saber el número de elementos, tendremos un grid consistente.

main{

height: 100vh;

display: grid;

grid-gap: 10px 30px;

grid-template-columns: repeat(3, 1fr);

grid-template-rows: repeat(3, 1fr);

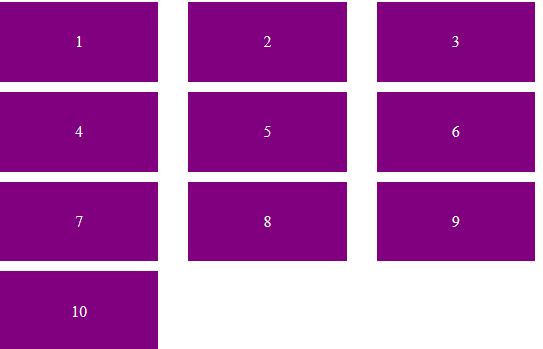
*/\*\**

*\* grid-template-columns: auto auto auto;*

*\*/*

grid-auto-rows: 1fr;

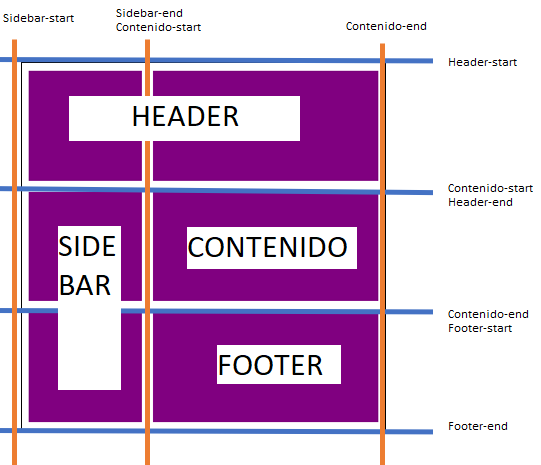
}



## Lección 04 Y 05 – Usando Nombres GRID Lines START Y END

Para las siguientes dos lecciones, se usará nombres para cada una de las lineas en base al siguiente recuadro el cual es:

* Un grid de 3 filas y 2 columnas.
* La segunda columna ocupa 2fr y la primera solo 1fr.



Los elementos que ubicaremos vienen indicados en el siguiente HTML:

<body>

<main>

<header>Header</header>

<aside>Sidebar</aside>

<article>Contenido</article>

<footer>Footer</footer>

</main>

</body>

Empezamos por definir el grid container, el cual será el elemento *main* junto a los template de filas y columnas respectivamente:

* Como se observa en corchetes va el nombre (o nombres) que recibirá la Grid Line y seguido su tamaño. La última linea no requiere de un tamaño.

main{

*display*: grid;

*height*: 95vh;

*grid-gap*: 10px;

/\* grid-template-columns: 1fr 2fr; \*/

*grid-template-columns*:

[sidebar-start] 1fr

[sidebar-end contenido-start] 2fr

[contenido-end];

*grid-template-rows*:

[header-start] 1fr

[header-end contenido-start] 2fr

[footer-start contenido-end] 1fr

[footer-end];

}

Y acomodando cada uno de los elementos, se le asigna desde que Grid Line hasta cual va, todo basado en la imagen principal de la lección.

header{

*grid-column*: sidebar-start / contenido-end;

}

aside{

*grid-row*: contenido-start / footer-end;

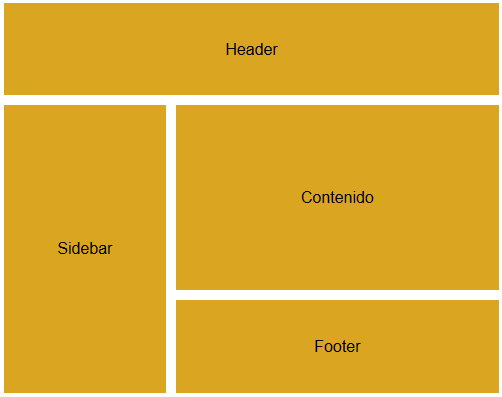
}

footer{

*grid-column*: contenido-start;

}

El resultado obtenido sería:



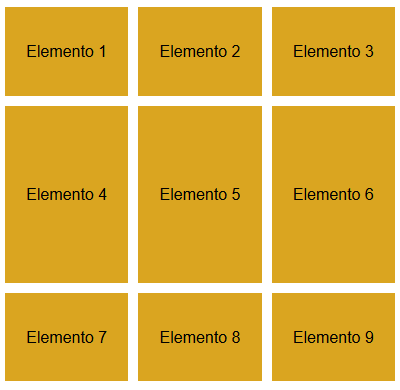
## Lección 06 – Unidad FR

Al usar *fraction* se divide el espacio **disponible** en el número de fracciones indicadas.

*grid-template-columns*: 1fr 1fr 1fr;

*grid-template-rows*: 1fr 2fr 1fr;

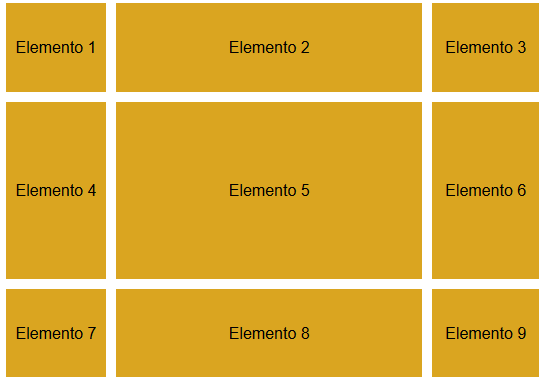
En este caso (*grid-template-rows)*, se reparte en 4 pedazos el espacio. (1+2+1 = 4). En el ejemplo de la lección, esto se veria de la siguiente manera:



Las unidades de medida para las filas y columnas se pueden combinar, es decir se puede usar *fracciones, pixeles, porcentajes, etc.* Lo importante es recordar que se reparte el espacio primero con las unidades normales y el **espacio disponible restante**será para las fracciones. Por ejemplo:

*grid-template-columns*: 100px 1fr 20%;

La primera columna toma 100 pixeles, la tercera columna el 20% del ancho disponible y, por último, por **ser el espacio disponible restante**, la fracción ocupa todo este espacio de la mitad.



Haciendo media queries, fácilmente podemos cambiar las dimensiones del Grid:

* Max-width: 600px
  + 2 columnas
  + 5 filas
* Max-width: 400px
  + 1 columna y 9 filas

@media (*max-width*: 600px){

main{

*grid-template-columns*: 1fr 1fr;

/\*grid-template-rows: 1fr 1fr 1fr 1fr 1fr;\*/

*grid-template-rows*: repeat(5, 1fr);

}

}

@media (*max-width*: 400px){

main{

*grid-template-columns*: 1fr;

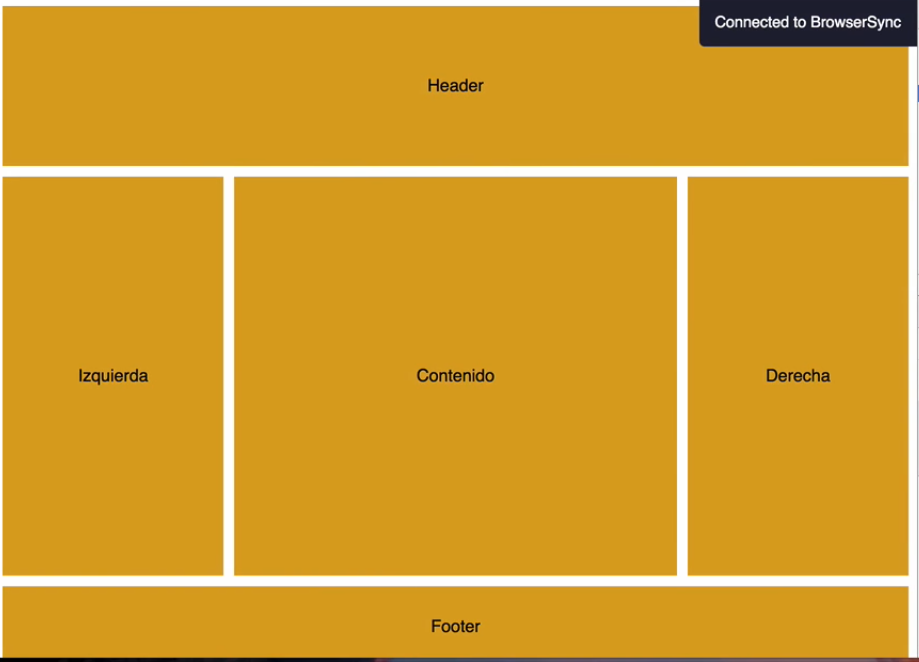
*grid-template-rows*: repeat(9,1fr);

}

}

## Lección 07 – GRID Template Areas

En la primera lección se usa los template areas. Los template areas son totalmente compatibles con las propiedaes de ***grid-template-columns y rows***



## Lección 08 – Establecer límites con minmax()

*Minmax*() sirve para delimitar las proporciones de las columnas asignando un tamaño minimo y máximo. Por eso se le conoce como un limite. En el siguiente ejemplo, se le asigna un mínimo de 100 px y un máximo de 1 fr a las 8 columnas.

*grid-template-columns*:

/\* repeat(8,1fr) \*/

/\*Designando un limite inferior y superior de las columnas\*/

/\*8 columnas con un limite: minimo: 100 px - máximo: 1fr\*/

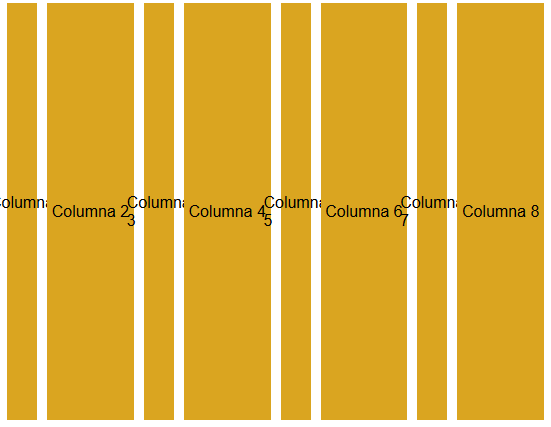
repeat(8, minmax(100px,1fr))

;

También se puede hacer patrones con la función *repeat* (tratada en detalle más adelante en otra lección)

/\*8 columnas con un Patrón de 2: 30px primer columna, minmax(10%,1fr) \*/

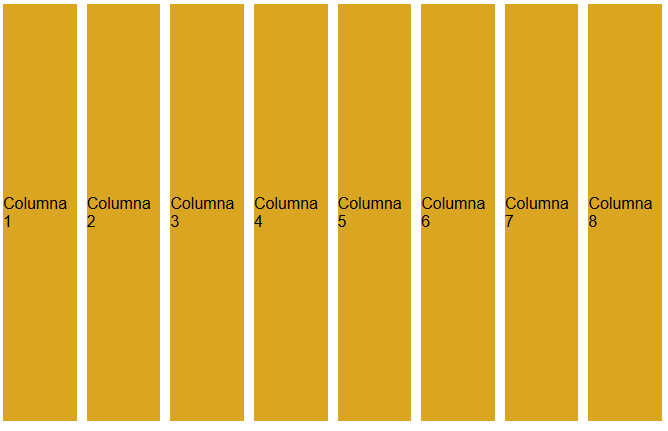
repeat(4, 30px minmax(10%,1fr))



También se puede dar valores como minimo y máximo de acuerdo al contenido:

/\*8 columna: minimo por contenido y máximo por contenido. \*/

repeat(8, minmax(min-content, max-content))



## Lección 09 – Repetir valores y patrones con repeat()

Método repeat se usa para crear varias columnas ya sean por valores o patrones.

* Crear un patrón que de como resultado 12 columnas y que el patrón vaya creando de a tres columnas con las siguiente caracteristicas:
  + 80 px y 1 fr
  + 40px y 1 fr
  + 20 px y 1 fr

Asi que como el patrón crea de a tres columnas, se repiten solo 4 veces.

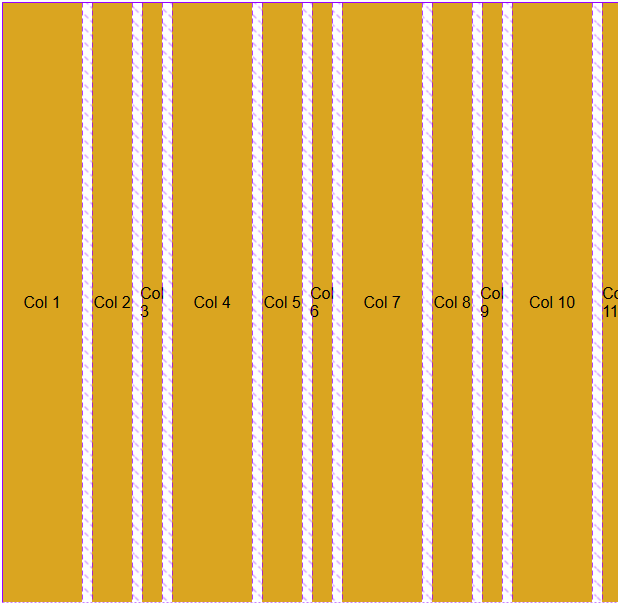
repeat(4,

minmax(80px,1fr)

minmax(40px,1fr)

minmax(20px,1fr)

)



* También dentro de los repeat, se puede nombrar cada una de las lineas:

[start]

repeat(4,

*[col-lg-start]* minmax(80px,1fr)

*[col-lg-end* *col-md-start* *]* minmax(40px,1fr)

*[col-md-end* *col-sm-start]* minmax(20px,1fr)

)

[end]

Esto sirve principalmente para referenciarlas de nuevo (Como se muestra en la lección 04 y 05) desde los elementos e indicar como se expanden dentro de la rejilla, por ejemplo:

Indicando que el primer ***div*** se ubique desde la *col-lg-start* 1 (el número indica el nth del nombre de la columna) hasta la segunda vez que se encuentra la columna con el nombre *col-mg-end*

div:first-of-type{

*background-color*: green;

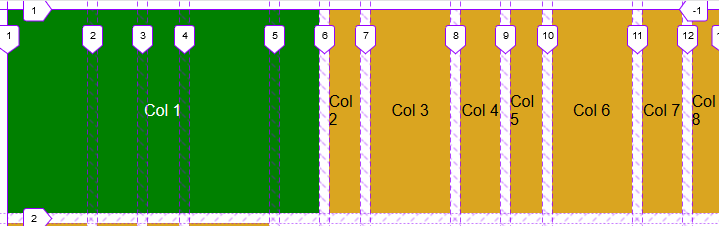
*color*: white;

*grid-column*: col-lg-start 1 / col-md-end 2;

*you can use a number to tell the layout “find the nth occurrence of the named line\**

}

Esto generaria un resultado similar:



## Lección 10 – GRID dinámico con auto-fill y auto-fit

Auto fill y auto fit son valores que también se pueden usar con *repeat().* Son unas propiedades bastantes útiles porque permiten que haya un *wrapping* de las columnas sin tener que asignarles un valor fijo.

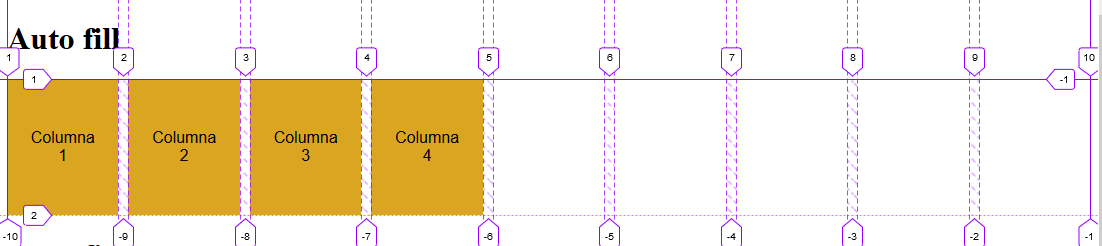
Aunque parezcan terminos opuestos, auto fill y auto fit tienen casi las mismas caracteristicas:

* **AUTO-FILL:** Rellena la fila con tantas columnas como pueden caber. Crea columnas implicitas donde una columna pueda encajar, porque está tratando de rellenar toda la fila. Algunas columnas pueden estar vacias, pero siguen ocupando un espacio:

.auto-fill{

*grid-template-columns*: repeat(*auto-fill*, minmax(100px, 1fr));

}

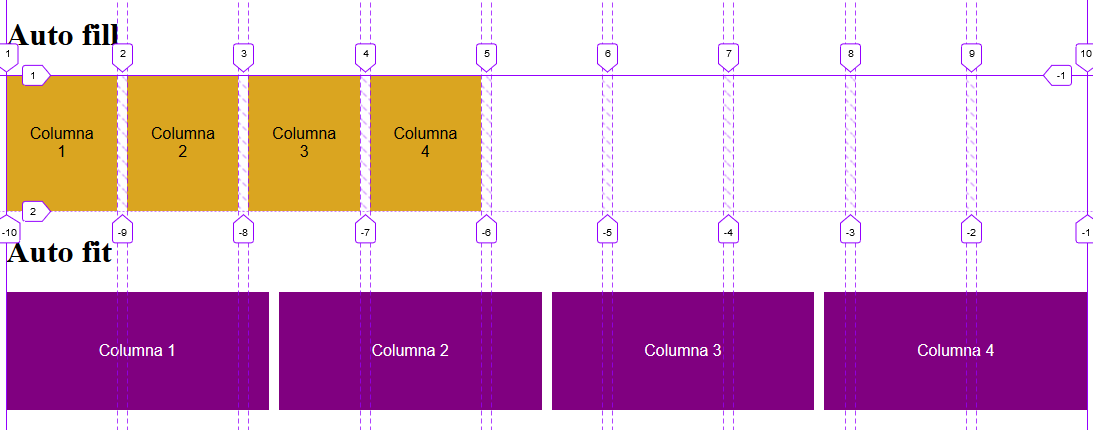


* **AUTO-FIT:** Encaja las columnas extendiendolas para que ocupen cualquier espacio disponible. En realidad, lo que ocurre es que primero se realiza el proceso de *auto-fill* y después el navegador toma el espacio vacio y colapsa las columnas disponibles para que ocupen este espacio. Tiene un poco de más sentido viendo la comparación directa con *auto-fill*.

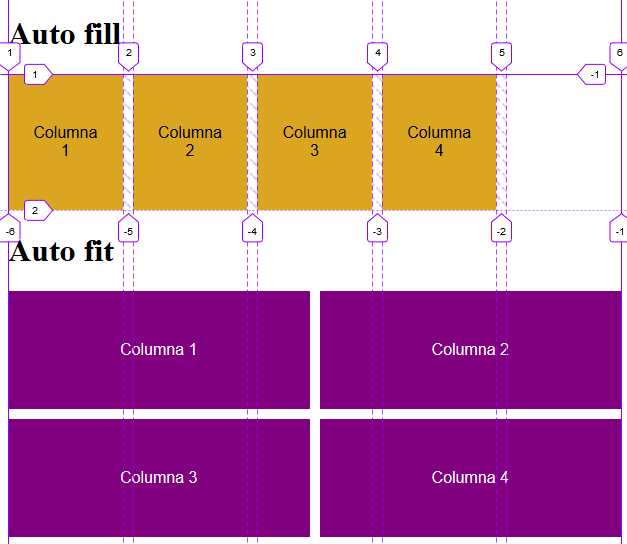
.auto-fit{

*grid-template-columns*: repeat(*auto-fit*, minmax(200px, 1fr));

}



Otro ejemplo, con el viewport más reducido:



Estos comportamientos se evidencian únicamente cuando la fila es lo suficiente ancha para encajar más columnas en ella.

## Lección 11 y 12 – GRID Explicit e Implici, auto-placement y packing modes.

Auto-placement

En un escenario de **300 elementos**, donde el contenedor tiene un ancho y alto del 100% del *viewport*, se ha creado las siguientes columnas y filas:

main {

*width*: 100vw;

*height*: 100vh;

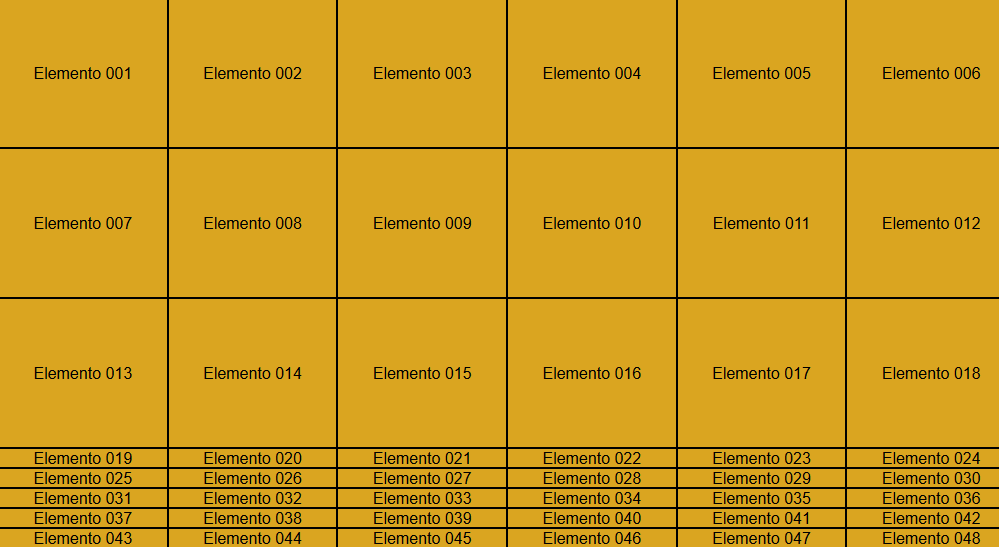
*display*: grid;

*grid-template-columns*: repeat(*auto-fit*, minmax(150px, 1fr));

*grid-template-rows*: repeat(*auto-fit*, minmax(150px, 1fr));

}

Sin embargo, al tener 300 elementos, se puede observar que la primera parte, la que hemos definido *(explicit grid)*, ocupa el 100% del viewport pero el resto de los elementos (que no caben en el tamaño del *main*) comienzan a ubicarse de manera implícita *(implicit grid)* porque no le hemos dicho a CSS Grid qué hacer con ellos.



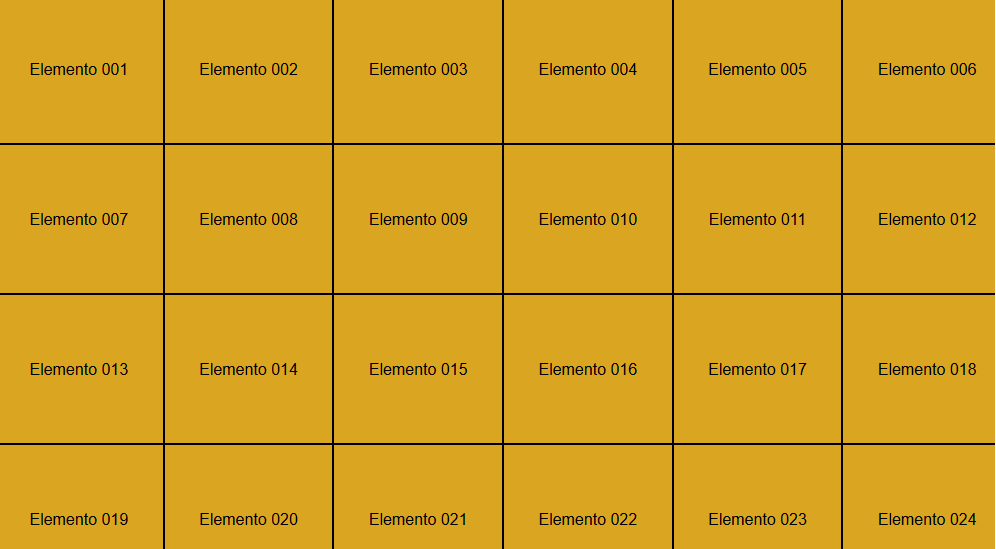
* **Explicit Grid:** Elementos ubicados de acuerdo a mis indicaciones y a las dimensiones o limites que se han establecido.
* **Implicit Grid:** Elementos que no se les ha dicho nada, empiezan a ubicarse de manera *auto.*

**¿Entonces qué hacer con el Implicit Grid?**

Lo que la siguiente propiedad hace es lidiar con las filas creadas a partir del *implicit grid* de manera *auto*.

*grid-auto-rows*: minmax(150px, 1fr);

El resultado es el siguiente:



Por supuesto, no tienen que ser las mismas medidas de los elementos del *explicit grid.* También, cabe mencionan que existe la misma propiedad para columnas, pero se debe cambiar el **Flow del Grid** para entenderlo.

Flow del Grid

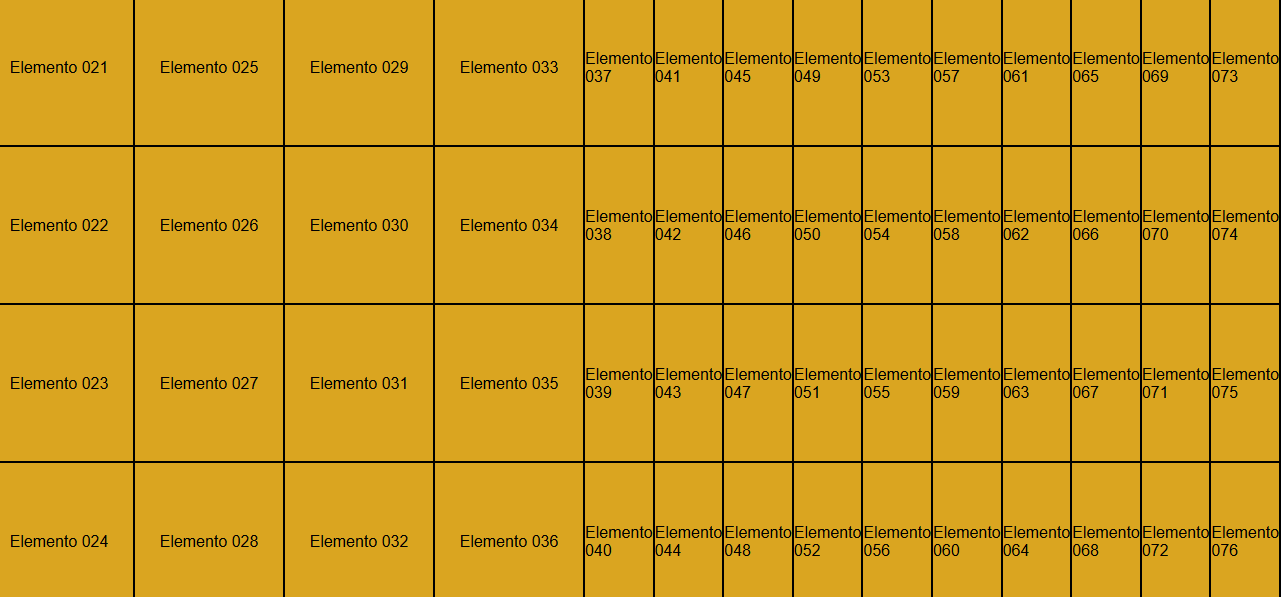
Los elementos del CSS Grid vienen normalmente con el valor de filas

*grid-auto-flow*: row;

Si se cambia a:

*grid-auto-flow*: column;

El resultado normal sería:



Y aquí se podríamos aplicar la propiedad de *grid-auto-columns* para que los elementos implícitos se acomoden a las medidas que deseemos.

Packing modes

Añadiendo la siguiente parte al CSS, podemos observar que se está indicando que cada ciertos 3, 7 y 12 elementos, se apliquen propiedades en especifico que incluyen extenderse en el grid, cambiar el color, etc.

section:nth-child(3n) {

*background-color*: purple;

*grid-column-end*: span 2;

}

section:nth-child(7n) {

*background-color*: cyan;

*grid-row-end*: span 2;

}

section:nth-child(12n) {

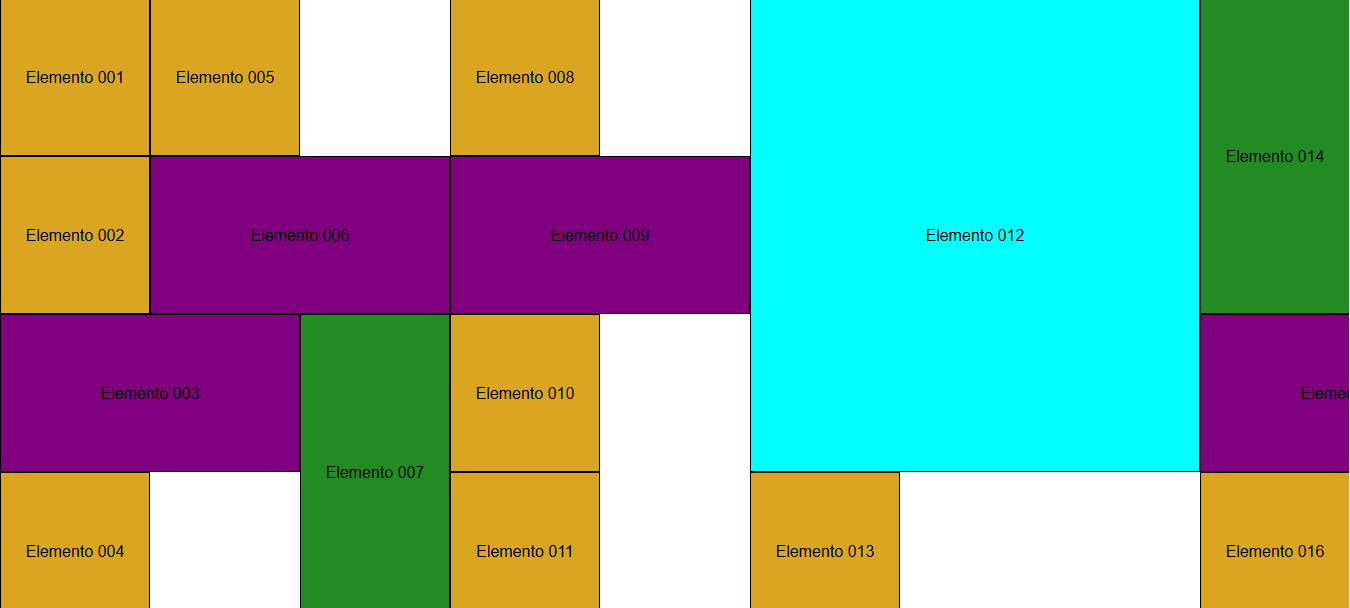
*background-color*: aqua;

*grid-column-end*: span 3;

*grid-row-end*: span 3;

}

**Y el resultado normal, es un poco desordenado:**

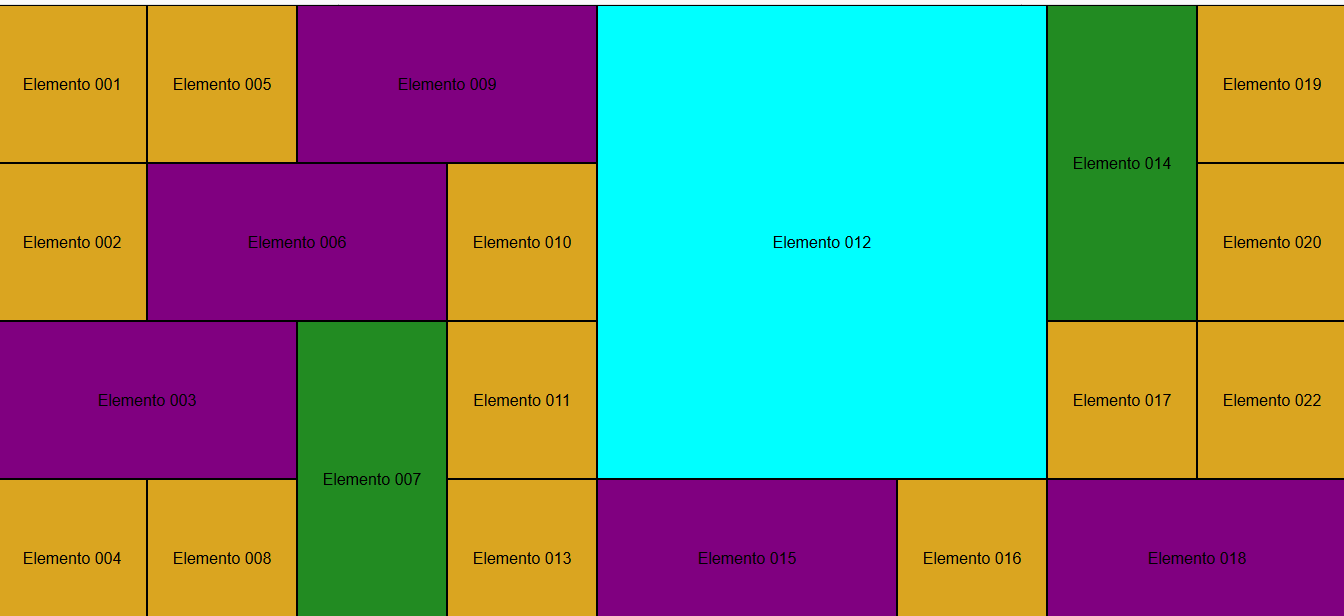


Lo que se quiere dar a entender es que se pueden notar varios *gaps o huecos* dejados porque es la propiedad por defecto y no altera el orden establecido en el DOM de forma visual. Como se observa se mantiene el orden visual. **(El Flow está establecido de columnas).**

No obstante, hay una forma de rellenar esos huecos si establecemos el packing mode dense:

*grid-auto-flow*: column dense;

**Y el resultado es:**



Cabe resaltar que el orden en el DOM de está manera no cambia*,* pero el orden visual si lo ha hecho y no corresponde.

## Lección 13 – Re ordenar Grid ítems con la propiedad order

Todos los elementos empiezan con un valor de 0 en la propiedad *order.* Si se asigna un orden de un número mayor se ubicará después del 0. También se puede usar números negativos.

div:nth-child(4){

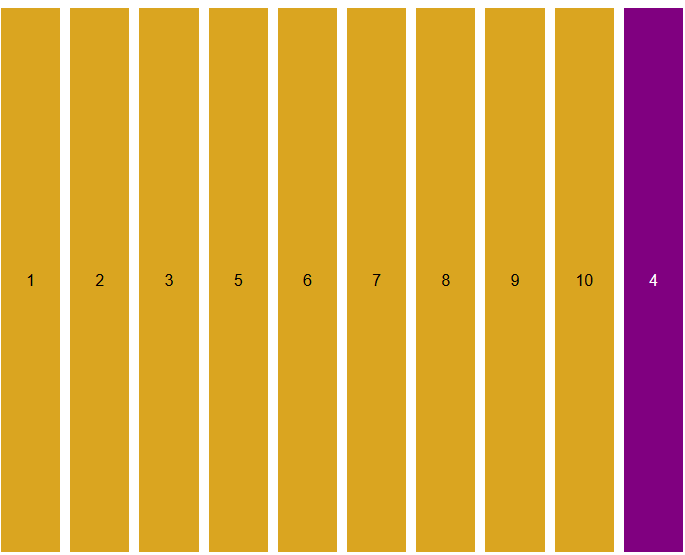
*background-color*: purple;

*color*: white;

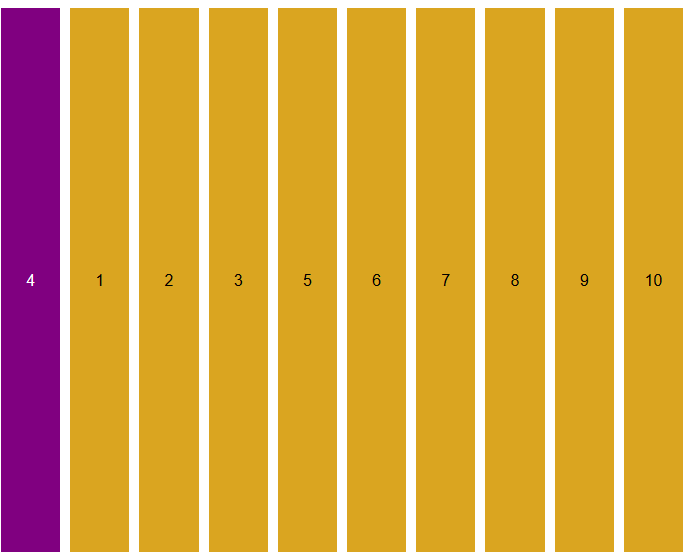
*order*: 1;

}

* Usando **order** = 1



* Usando **order** = -1



## Lección 14