# Desarrollo web: Clase 0

#### ¿Qué es el desarrollo web?

Es la planificación y diseño de paginas de internet.

* Principios básicos:
  + Navegabilidad: A donde se puede ir dentro de la misma página.
  + Interactividad: Es la manera en que se pasa de un método a otro. Es la manera en la que nos relacionamos con la página. Se incluyen botones, formularios, etc.
  + Arquitectura de la información: Es la manera en que se organiza nuestro sitio web.
* Herramientas básicas del desarrollo web
  + HTML: Es un lenguaje de marcado de etiquetas, de hipertexto. Todo lo que se ve en la web son archivos html. Se encarga solo de la parte estructural, no del diseño en sí.
  + CSS: Permite controlar la apariencia de una página web. Son hojas de estilo en cascada.

#### Conceptos básicos sobre Internet

* Se trata de una red de equipos que se encuentran interconectados para compartir información. Es donde nosotros nos conectamos.
* Navegador: es la manera en la que entramos a internet. Es un software.
* Buscador: sistemas de información que nos permiten buscar cosas en internet.

#### Que es un sitio web?

Es un consunto de archivos y directorios que están almacenados en un servidor web, que está conectado a internet.

Una página web es un documento que se puede mostrar en navegadores web, mientras que un sitio web es una colección de páginas web.

Todos los sitios web deben tener un archivo llamado index.html, que será la página que se abrirá por defecto cuando se entre al sitio web.

El contenido de la pagina siempre se escribe dentro de body.

* Titulos:
  + **<h1>**: Es el titulo de primer nivel.
* Párrafos
  + **<p>**
* Etiqueta titulo: Va dentro del head. Es el título que aparece en la pestaña del navegador**.**

# Clase 1: prototipado y conceptos básicos de HTML

Para iniciar un proyecto web, debemos tener un bosquejo de que es lo que queremos hacer

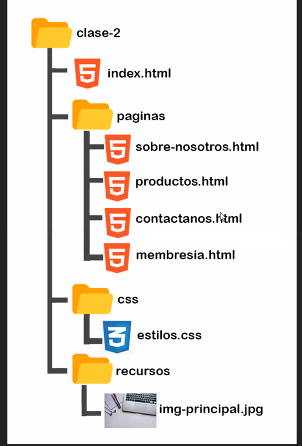
Prototipado: Poder plasmar nuestra idea de manera visual. Fases:

* Sketch: Primer dibujo de la página que se quiere hacer. Reproduce de manera sencilla una idea.
* Wireframe: se tiene que tener bien establecido los lugares que va a ocupar la página. Es un proyecto de baja calidad que nos permite conocer el:
  + ¿Que?: Los principales grupos de contenido.
  + ¿Como?: La descripción y visualización básica del usuario.
  + ¿Donde?: La estructura de la información.
* Mockup: es como un wireframe, pero con una calidad aumentada. Permite la visualización del contenido y demuestra las funcionalidades básicas de una manera estática, permite revisar la parte real de un proyecto.
* Prototipo: Es como si la pagina estuviera funcionando. Se puede interactuar con la página.

En este curso se llegará solo hasta la parte de wireframe.

Arquitectura del proyecto: se compone por la carpeta raíz y diferentes archivos. En estos nos damos cuenta cuales son las carpetas o directorios. Se crean las siguientes carpetas:

* resources: Van todos los recursos del proyecto (imágenes, videos, etc.). Formato de las imágenes: jpg generalmente, salvo que se tenga un fondo transparente en donde se tiene que usar png.
* css: van los archivos de estilo
* pages: van todas las páginas adicionales al index.html. Este último va dentro de la carpeta raíz.



#### HTML

Es un lenguaje de marcado de etiquetas para crear documentos web. Permite indicar donde queremos cada elemento. Solo se encarga de la parte estructural, no del diseño.

Convenio de nombre de archivos:

1. No deben tener espacios, acentos, eñes, símbolos ni espacios.
2. Deben estar escritos todos en minúsculas
3. Deben tener la extensión .html.
4. Se pueden usar números, pero es preferible evitarlos.
5. Aplica tanto para archivos como para carpetas.

Documento predeterminado

Es el archivo que se muestra cuando se pide una URL en un browser. Por convención, se debe llamar index.html.

Existen dos tipos de etiquetas en html:

* Etiquetas cerradas. Hay una etiqueta que abre, y otra que cierra, las cuales encierran un contenido, por lo general texto. Ej: **<p></p>.**
* Etiquetas abiertas: Estas no cierran contenido, se valen por si solas. Ej: **<hr>**.

Las etiquetas pueden tener atributos. Estas se colocan en la etiqueta de apertura.

* Comentarios:
  + **<!—COMENTARIO -->**: No se ve en la página web, pero si se ve en el inspector de la página.

Etiquetas generales:

* Identificación de el tipo de documento (HTML)
  + **<!DOCTYPE html>:** Declara el tipo de documento.
* Etiqueta html
  + **<html> </html>**: Va todo nuestro código.
    - Atributo **lang=””**: Identifica el idioma en que se va a cargar la página.
* Etiqueta head
  + **<head></head>**: Es la parte privada del documento, la que no ve el usuario. Envuelve otras etiquetas importantes como **<title>,** las etiquetas **<meta>** y las relacionadas con el importe de documentos CSS y JS.
* Etiquetas meta: Establecen diferentes configuraciones que va a tener el navegador
  + **<meta charset =” UTF-8”>** Establece los caracteres que se van a utilizar. UTF-8 es el tipo de caracteres que se utiliza en español e inglés.
  + **<meta http-equiv="X-UA-Compatible" content="IE=edge">** Es para que los navegadores con internet explorer funcionen bien.
  + **<meta name="viewport" content="width=device-width, initial-scale=1.0">:** Define el tamaño de la pantalla.
* Etiqueta de títulos:
  + **<title></title>:** Va el título que aparece dentro de la pestaña
* Etiqueta de cuerpo:
  + **<body></body>** va todo el contenido de la página que se puede ver.
  + Las etiquetas dentro de body se dividen en dos grupos:
    - Elementos de bloque: Son los que ocupan el 100% del ancho de su contenedor. Se muestran uno debajo del otro.
    - Elementos de línea: Solo ocupan el ancho que diga su contenido. Se ven uno al lado del otro.
* Títulos (header):
  + **<h1> </h1> a <h6> </h6>**. Se recomienda poner solo un h1 por página. Se utilizan de manera jerárquica. Generalmente se utilizan los h1 a h6. Es solo para títulos, no se debe escribir contenido con las etiquetas h. No debes tener un **<h3>** si antes no hubo un **<h2>**. Además, están relacionados: Cada **<h3>** debe ser un subnivel de contenido del **<h2>** inmediato-anterior. Y si dos títulos son igual de importantes, son el mismo nivel de encabezado. Y si el título no es igual al anterior (ni tampoco es un sub-nivel) se volverá al nivel de encabezado anterior.
* Etiquetas de párrafo (paragraph):
  + **<p>** **</p>:** Es una etiqueta que se usa para escribir largas líneas de texto. No puede haber un párrafo dentro de otro párrafo.
* Texto en negrita (black Font):
  + **<b></b>**.
* Texto en cursiva (italics):
  + **<i></i>**
* Letra chica (small):
  + **<small></small>**
* Strong:
  + **<strong></strong>:** etiqueta utilizada para marcar con especial énfasis las partes más importantes de un texto.
* Salto de línea (break):
  + **<br>.** Inserta un salto de línea simple. Son para separar oraciones, no un párrafo de otro. Se encuentra esta etiqueta en desuso, no se la debe utilizar.

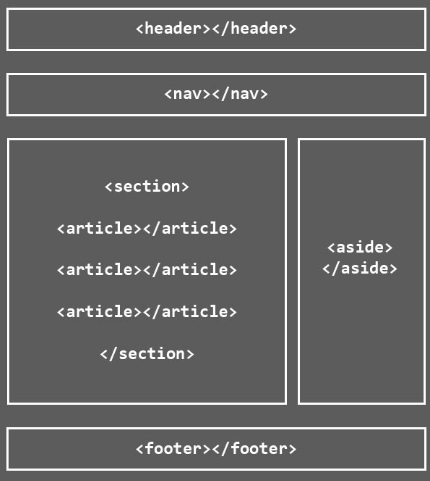
Etiquetas de usos múltiples

Son utilizadas principalmente para dar formato en CSS y orden.

* **<div></div>** Es un divisor, y se comporta como un elemento de bloque. Se puede incluir dentro de cualquier elemento HTML.
* **<span></span>** Es un divisor, y se comporta como un elemento de línea.

Nuevas etiquetas HTML 5 (HTML Semántico)

HTML 5 incorpora etiquetas semánticas que generan estructura y definen su contenido.



En la parte superior de una página web deben encontrarse un encabezado y/o un panel de navegación, debajo se encuentra una área denominada artículo conformada por distintas secciones y al lado el aside (barra lateral). Por último se agrega un footer, o pie de página.

* **<nav> </nav>:** Es para establecer la navegación de nuestra página web. Contiene los enlaces de navegación, como por ejemplo el menú.
* **<section></section>**: Se usan para separar contenidos. El main es el único que se puede dividir en sections. Define una sección en el documento.
* **<article></article>**: Define contenido autónomo que podría existir independientemente del resto del contenido.
* **<aside></aside>:** contenido extra en una pequeña sección que suele estar a un lado de la página.
* **<footer></footer>:** Se utiliza para agregar copyright, redes de contacto, etc.

Las etiquetas se pueden anidar a otras etiquetas. Esto quiere decir meter una etiqueta dentro de otra. Es importante que siempre se cierran en orden inverso al de la apertura.

Las paginas web tienen tres grandes secciones:

* **<header> </header>** es el encabezado de la pagina web. Incluye la barra de navegfacion y el logo.
* **<main>** va todo el contenido de la pag que no sea header o footer. Va el contenido principal del documento.
* **<footer>** es el pie de página. Van los datos de contacto, los derechos reservados, etc.

Etiquetas semánticas: son las que reconoce el navegador para identificar las secciones. No tienen otra función concreta. Nos dice que lo más importante debe estar en el main.

El header y el footer deben ser exactamente iguales en todas las paginas que diseñemos, esto es por patrón de diseño.

# Clase 2: Primeros pasos con HTML

#### Enlaces

Son la clave para la navegabilidad del sitio.

Se manejan desde nuestra posición hacia el lugar donde nosotros queramos ir. Los enlaces se ponen entre comillas.

Para moverme entre carpetas, si quiero entrar a una carpeta que este dentro, tengo que usar la barra /. Si quiero volver para atrás, debo poner dos puntos y luego la barra ../. Estos dos puntos quieren decir que se vuelve un solo directorio hacia atrás.



Si estoy en el index, y quiero ir al archivo membresías: “paginas/membresia.html”.

Si estoy en membresía y quiero volver al index: “../index.html”.

Si estoy en membresía y quiero ir a productos: “productos.html”.

La carpeta raíz no se la debe nombrar.

Los enlaces se declaran como:

* **<a href=”pages/contacto”></a>**

Dentro del atributo href, se puede poner un #, que hace que se recargue la página. Es una buena práctica ponerlo dentro del index.

Las navegaciones <nav­­> se hacen siempre con listas.

#### Listas

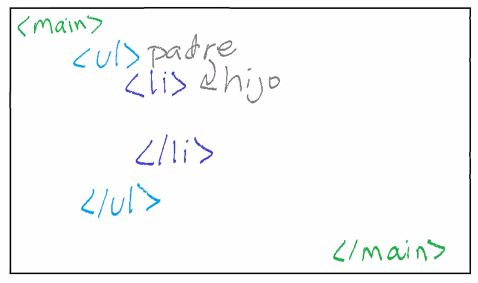
Hay dos tipos de listas: ordenadas (enumeradas) y listas desordenadas (tienen viñetas).

* Lista ordenada:
  + **<ol></ol>**
* Lista desordenada:
  + **<ul></ul>**
* Cada uno de los elementos que van dentro de las listas se ponen con el elemento li. Cada elemento va dentro de un li diferente.
  + **<li></li>**

Las listas ordenadas se usan generalmente para seguir pasos. Las navegaciones generalmente se hacen con listas desordenadas.

#### Etiqueta padre y etiqueta hijo

Dentro de main tengo ul, y dentro de ul tego diferentes li. ul es la etiqueta padre de li, y ul es etiqueta hijo de mail. No existen las etiquetas abuelo. Las etiquetas padre pueden tener múltiples hijos.



#### Imágenes

Existen dos formas de colocar las imágenes:

* Enlace local
  + **<img src=”resources/laptop.jpg” alt=”Texto alternativo a la imagen”>**
* Enlace web
  + **<img src**

La etiqueta imagen tiene un atributo **source**, que nos indica la ruta de la imagen.

Atributo **alt:””,** es para poner un texto alternativo a la imagen que se mostrará cuando no se cargue la imagen. Conviene poner este atributo ya que sube el posicionamiento web.

No se debe utilizar los atributos heigth o width, ya que va en contra del SEO.

#### Videos

Se recomienda que los videos se pongan directamente de youtube, ya que si no va a tardar mucho la página en cargar.

Para cargar un video de youtube, debo entrar al mismo, y ver la opción compartir y dentro de esta incorporar. Esto nos va a traer una etiqueta **<iframe>**, que se usa para abrir pedazos de otra página en la nuestra.

En esta etiqueta iframe, se puede usar el heigth y width.

Iframe se puede utilizar para otras páginas, como Google maps o twitter.

#### Audios

Etiqueta en desuso, ya que a nadie le gusta que una página tenga sonido por si sola.

#### Primer desafío: wireframe y estructura del proyecto

No es necesario incluir el formulario ya que no se vio en clases. El Wireframe es solo de la página del index. La página web debe cumplir con la arquitectura.

Se tienen que entregar todos los html.

# Clase 3: Incluyendo CSS a nuestro proyecto

Ya no se usan tablas en las páginas web.

* **<div></div>:** Es una etiqueta para dividir contenido. Es un contenedor divisor. Se divide el contenido (por ejemplo, de un formulario) en contenedores. No se debe utilizar la etiqueta <br>, es una mala práctica. Se lo utiliza para seccionar el contenido.

#### Formularios

Se utilizan dentro de muchas fases, en especial dentro del área de contacto.

* **<form></form>:** Es la etiqueta para formularios
  + Atributo **method=””**: puede ser
    - **get:** la información viajará por la barra de direcciones a continuación del nombre del archivo. Va por ejemplo el color de una remera. No debería ir el mail del usuario o el tipo de tarjeta de crédito.
    - **post**: la información viajará junto a los encabezados de HTML (será invisible). Todos los datos sensibles deberán ser post.
    - Si no se indica el método, por defecto será **get**.
  + atributo **enctype**: es para comunicarse con el backend.
  + atributo **action**: es para trabajar con javascript. Nos indica a donde se mandarán los formularios una vez estos sean validados.

Los parámetros se colocan en input y textarea

* **<input>**: Textos de solo una línea. Es una etiqueta que no cierra.
  + Atributo **type=””**. Puede ser:
    - **submit.** Botón utilizado específicamente para enviar los datos del formulario.
    - **reset**
    - **buttom**. Es para agregarle una funcionalidad con javascript. Va acompañado siempre del atributo **value=””**.
    - **checkbox:** casillas de chequeo
    - **radio:** es un checkbox redondo.
    - **text**
    - **number:** solo permite introducir números.
    - **password:** se ingresa texto que se tapa con \*.
    - **color:** permite elegir un color desde una paleta.
    - **range:** barra deslizable. Permite ingresar los atributos **min=””** y **max=””.**
    - **date:** formato dd/mm/aaaa
    - **time**: formato hora.
  + Atributo **value=””.** Es lo que dice que va a hacer el input.
  + Atributo **name=””**.
  + Atributo **placeholder=””**. Se utiliza para mostrar texto indicativo que desaparecerá al ingresar el primer carácter.
* **<label></label>:** Es el texto que va justo al lado del input. Su principal atributo es **for=””**, que va a referenciar a **label** con su elemento del formulario. El valor del atributo “for” debe ser igual al valor del atributo “id” o “name” del elemento.
* **<textarea></textarea>**: Texto que amerita más de una línea, nunca información sensible.
* **<select></select>:** Es para poner un menú desplegable.
  + **<option></option>**: Van los elementos del menú desplegable.

#### Favicon

Es el ícono pequeño que aparece al lado del nombre de la pestaña. Para el favicon se utiliza generalmente el .ico. Para ponerlo, se pone dentro del head:

* **<link rel="shortcut icon" href="favicon.ico" type="image/x-icon">**

## CSS

Los archivos css se llaman sí o sí estilos.css o styles.css.

Se encarga solamente de cargar los estilos de la página.

#### Sintaxis:

**selector{**

**propiedad1:** valor;

**propiedad2:** valor;

**}**

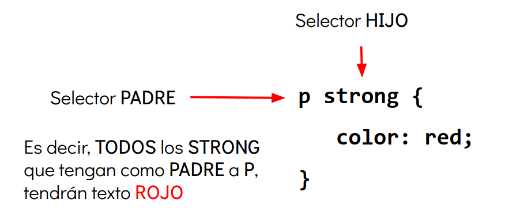
Es importante poner siempre los punto y coma dentro del código.

#### Reglas sintácticas

1. Cada declaración en CSS está formada por un juego de pares **propiedad:valor**, separado por los dos puntos. Las propiedades se separan entre sí con los punto y coma.
2. Las propiedades no se ven afectadas por los espacios en blanco. Se pueden escribir de corrido o una debajo de la otra (así es más legible).
3. Siempre que la propiedad represente un número, el valor debe indicarse en que unidad se expresa. Entre el número y la unidad no pueden existir espacios.
4. Los colores se pueden expresar de tres maneras distintas:
   1. Por el nombre del color en inglés,
   2. por su ccombinaciñon hexadecimal (numeral más 6 números),
   3. Por rgb(red,green,blue), tres números de 0 a 255 separados por coma.

#### Padres e hijos

A la hora de crear las diferentes clases en CSS, se puede agregar atributos específicos a los hijos sin alterar los atributos del padre. Esto se hace dejando un espacio entre los selectores.



#### Insertar CSS en HTML

Se puede dar formato CSS de diferentes maneras:

* Todas las etiquetas HTML tienen el atributo **style=””**, donde entre las comillas se escribirán las reglas CSS para dar formato únicamente a ese elemento.
* Existe una etiqueta **<style></style>** que va en el **<head>** y contendrá las reglas CSS para aplicar únicamente al archivo HTML donde se haya insertado.
* Utilizar la etiqueta **<link>** que va dentro del **<head>** y se la usa para cargar un archivo externo .css. Esta forma permite dar formato a múltiples archivos HTML.
  + <link rel="stylesheet" href="css/styles.css">

Importante: Herencia de etiquetas.

Todas las etiquetas HTML se las pueden utilizar como selectores.

Tiene que existir solo un único archivo CSS para todas las páginas de nuestra web.

#### Propiedades:

* Estilos de texto:
  + **color: blue;** cambia el color del texto de la etiqueta.
  + **font-size: 16px;** Generalmente los tamaños de las letras se ponen en pixeles.
  + **text-align: center**; Se usa para alinear texto (solo texto!!)
  + **font-family: ;** Ver como se cargan desde Google Font. Para el contenido, como estándar se utiliza la letra roboto. Para los títulos se puede utilizar otra. Se recomienda utilizar directamente las letras desde la api de Google.
  + **text-decoration: none;** es para darle el subrayado.
  + **font-style: italic;** muestra el texto en itálica.
  + **font-weight: bold;** muestra el texto en negrita.
  + **letter-spacing: ..px;** espaciado entre letras.
  + **word-spacing: ..px;** espaciado entre palabras.
  + **line-height: 1.6;** es para darle altura del interlineado
* Estilo Background
  + **background-color: red;** se usa para darle color al fondo del texto.
  + **background-image: …;** se usa para poner una imagen de fondo. No demasiado utilizado, ya que dificulta la lectura.
  + **background-repeat: repeat-x;** Se usa para indicar como se debe repetir el background.
  + **background-position: posicionX posicionY;**
  + **background-size: ancho alto;**
* **width: 300px;** se usa para darle tamaño a una imagen.
* **heigth: 300px**; idem.

#### Selectores personalizados

Atributo en html:

* **class=””**
  + Para llamar el selector de clase en el CSS, debo colocar primero un punto.
  + No se deben crear dos clases que hagan lo mismo.
  + Se puede aplicar más de una clase a cada etiqueta HTML separadas por un espacio.
* **id=””**
  + No se debe utilizar los ID para dar estilos, estos se los reservan para dar funcionalidades con javascript.

#### Precedencia de declaraciones

Cuando reglas distintas apuntan al mismo objeto:

* Si son propiedades distintas, se suman (se combinan)
* Si alguna propiedad se repite, solo queda una

#### Unidades de medida

# Clase 4: CSS + Box Modeling

#### Metodología BEM

Es una manera de leer el código. Significa Modificador de Bloques de Elementos en inglés (Block Element Modifier).

BEM tiene como objetivo modulizar lo máximo posible cada uno de los bloques de código. Se centra en tres parámetros: los Bloques (div, section, article, ul, ol, etc), los Elementos (a, button, li, span, etc) y los Modificadores.

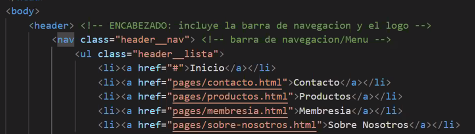
* El Bloque es un contenedor donde el elemento se encuentra presente. Es como si fueran partes estructurales de código más grandes. Puede que se tenga un encabezado, pie de página, barra lateral y un área de contenido principal. Cada una de ellas se la considera como bloque.
* El Elemento es una de las piezas que compondrán la estructura de un bloque. El bloque es todo, y los elementos son las piezas del bloque. En la metodología BEM, cada elemento se escribe después del bloque padre, usando dos guiones bajos.
* Cuando se necesite modificar el estilo de un elemento específico, se utiliza un modificador. Para ello, se añade un doble guión luego del elemento o bloque.

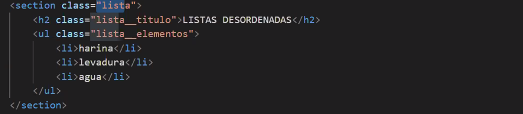
Los bloques se los puede considerar como la manera de anidar las etiquetas. Siempre habrá etiquetas dentro de otras etiquetas. El bloque podría ser la etiqueta de mayor jerarquía que envuelve a otros elementos dentro, por ejemplo, el header envuelve un logo un menú de navegación etc. representa al padre en ese anidamiento de etiquetas.

El único bloque grande que se subdivide es el main, el header y el footer no se subdividen. El main se segmenta en contenidos con el section.

BEM se usa para nombrar las clases. La nomenclatura es la siguiente:

* bloque\_\_elemento—modificador





#### Reseteo de CSS

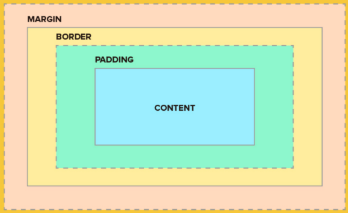
Al principio del css, se crea el siguiente código:



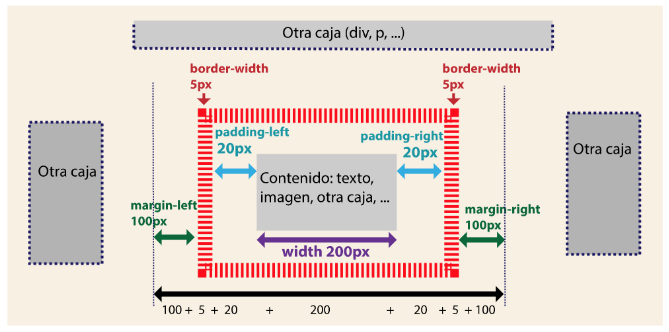
Siempre se debe incluirlas al principio de nuestro CSS.

#### Box Modeling

Permite definir los elementos por cajas. En HTML, todos los elementos son cajas.

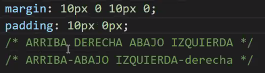


* Content: Es el espacio para texto o imagen. Ejemplo: Content sería el paquete.
* Padding: Es el espacio del contenido chocando contra el borde de la caja que lo contiene. Ejemplo: Padding es el papel burbuja que recubre un paquete dentro de una caja.
* Border: Es el límite entre el elemento y el espacio externo. Ejemplo: Border es la caja.
* Margin: Es el espacio entre la caja y otro elemento. Es el espacio entre contenedores. Ejemplo: margin es la distancia en la que están separadas las distintas cajas.



Margin y padding no se deben utilizar para centrar elementos.

Si no se aclara, el padding y el margin se aplicarán en todas las direcciones por igual. Si quisiera tenerlos distintos según la dirección, debo agregar 4 valores numéricos que definan el margin/padding según cada dirección (el primer valor es de arriba, el segundo derecha, tercero abajo y cuarto derecha). También puedo escribir dos números solamente, donde el primero representa el valor de arriba y abajo, y el segundo la derecha y la izquierda.



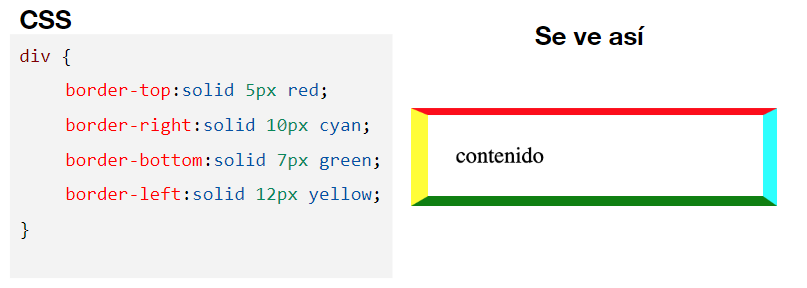
Se recomienda a todo contenido con texto agregarle padding, para que no queden pegados al borde y se dificulte su lectura (16 px de padding izquierda-derecha es el mínimo que se usa).

#### Border

Las propiedades border-top, border-right, border-bottom y border-left se utilizan para definir los bordes de cada elemento por separado. Se puede definir los 4 lados de forma abreviada poniendo solo border.

A diferencia de margin y padding, los bordes se forman contres valores:

* Tipo de borde (border-style)
* Grosor (-width)
* Color (-color)



#### Alto y Ancho

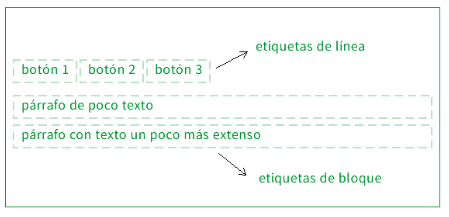
* width: Es la propiedad de CSS que controla la anchura de la caja de los elementos.
* height: Es la propiedad de CSS que controla al altura de la caja de los elementos.

#### Propiedades:

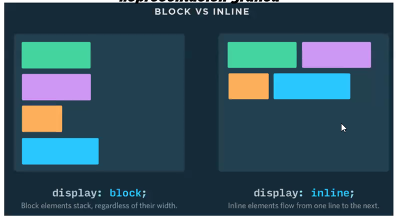
* list-style: none; Le saca los puntitos a las listas no ordenadas.
* display: inline/block;
* text-align: center; Se utiliza para centrar texto.
* text-decoration: none; es para hacer subrayados de texto. Si va none, se le quita el subrayado.
* background-image:url(‘’): Permite agregar una imagen de fondo como background
* overflow:…;
  + :visible; Valor por defecto, el excedente es visible.
  + :hidden; el excedente no se muestra (lo corta). Se recomienda utilizar este overflow.
  + :scroll; Se genera una barra de scroll en los dos ejes del elemento, aunque no se necesite.
  + :auto; genera solo scroll si es necesario.

#### Display

Recordar: Tipos de etiqueta de línea (se ven uno al lado del otro) y etiquetas de bloque (se ven uno debajo del otro).+



Hay etiquetas de bloque que queremos que se vean como etiquetas de línea y viceversa. Para poder intercambiarlas, entra en juego la propiedad en CSS llamada display



Recomendación para mantener orden: Ir poniendo los pares selectores-propiedades-valor en el orden en que van a apareciendo.



#### Posiciones

Es una propiedad de CSS pensada para ubicar un elemento. Al aplicar esta propiedad, se puede usar cuatro propiedades para posicionar los elementos, y debo darles un valor numérico: top, right, bottom y/o left.

Se utiliza un solo position actualmente. Se los considera hoy en día una mala práctica su uso. Solo en el curso vamos a usar dos:

* fixed: Se puede utilizar para crear elementos que floten, y se queden en la misma posición absoluta aunque se haga scroll.
* position:sticky; El elemento es posicionado en el Flow natural del documento, pero siempre queda visible. Hace que un elemento se “pegue” a la pantalla, y cuando se hace scroll para abajo quede fijo. Se queda fija la posición del elemento.

#### Medidas en CSS

Todas las medidas se indican como un valor numérico seguido de una unidad de medida, sin espacios en blanco entre el número y la unidad de medida.

* Unidades absolutas: in, cm, mm, pt, pc, px. Este último es el único que se utiliza frecuentemente.
* Unidades relativas:
  + em: tamaño de letra del elemento padre
  + rem: tamaño de la letra del elemento raiz
  + %: relativa a herencia, del elemento padre.
  + vw (viewWidth): Es el tamaño relativo de la pantalla por donde se está viendo. Es para manejar el ancho de la pantalla. 100 representa que se tomará el 100% del ancho de la pantalla.
  + vh (viewHeigth): idem vw, pero para el alto de la pantalla.

# Clase 5: Flexbox

Flexbox es un modo de diseño para crear estructuras para sitios web. Permite posicionar elementos en la posición que se desee, tanto en forma horizontal como vertical. También se puede establecer como se distribuirán, el orden que tendrán e incluso el tamaño que tendrán en proporción a otros elementos. Es muy útil para crear diseños adaptables a dispositivos móviles (responsive design).

Flexbox no es una propiedad ni conjunto de propiedades, es un nuevo modelo de layout que viene a incorporarse a los ya existentes en CSS. Un modelo de layout es un set de algoritmos que determinan el tamaño y posición de los elementos respecto a sus hermanos y ancestros.

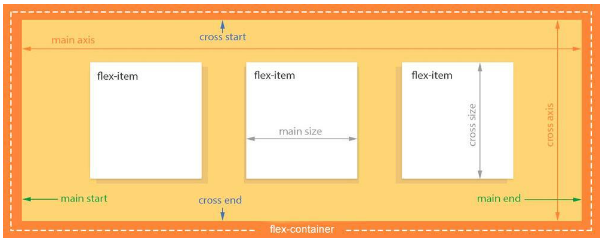
Con flexbox se puede hacer lo siguiente:

* Distribuir los elementos en sentido vertical u horizontal.
* Reordenar la aparición de los elementos.
* Ajustar dinámicamente para evitar overflows.
* Redefinir el sentido del flujo de los elementos (hacia arriba, abajo, izquierda o derecha).
* Alinear los elementos respecto al padre o los hermanos.

#### Conceptos básicos de Flexbox

La disposición de flexbox está constituida por elementos padres e hijos. El padre será el contenedor flexible (flex container) y los hijos directos (solo hijos directos, no nietos) serán los elementos flexibles (flex ítems).

El flex container se comportará como un elemento de bloque, sin embargo, los flex ítems modificarán su comportamiento.



Las propiedades de flexbox solo se aplican al padre, y el padre solo afecta a sus hijos directos. No hay abuelos, solo hay relación padre-hijo.

Flex se divide tantas veces como hijos tenga.

En una caja normal, el eje x es el main axis mientras que el eje y es el cross axis. A cada extremo del primero se lo denomina main-start (izquierda) y main-end (derecha), mientras que para el segundo son cross-start (arriba) y cross-end (abajo).

#### Propiedades para aplicar al flex container (al padre)

* **display: flex;** Es la propiedad que inicializa el flexbox. Se configura sobre el flex container, y sus hijos directos se convertirán en flex ítems de manera automática.
* **flex-direction:…;** Sirve para cambiar el main-axis. Esta propiedad va a permitir manejar el direccionamiento de los flex ítems. Nos permite especificar si queremos que los flex ítems se dispongan en filas o columnas.
  + **:row;** Mantiene el sentido del main axis. Es el valor por defecto. Los ítems se aplilan en una fila de izquierda a derecha.
  + **:row-reverse;** Invierte el sentido del main axis. Los ítems se apilan en una fila de derecha a izquierda.
  + **:column;** Gira el main axis en sentido horario. Los ítems se apilan en una columna de arriba hacia abajo.
  + **:column-reverse;** Gira el main axis en sentido antihorario. Los ítems se apilan en una columna de abajo hacia arriba.
* **flex-wrap:…;** Es para controlar los saltos de línea, modificando si queremos que los ítems puedan saltar de línea manteniendo la apariencia deseada. Fija el tamaño de las cajas, y frente a una disminución del viewport al excedente de las cajas lo pasa a una segunda línea.
  + **:nowrap;** valor aplicado por defecto, hace que los elementos encajen en el ancho del flex container aún modificando la apariencia de estos. Las cosas no se reorganizan según el tamaño de la pantalla.
  + **:wrap;** los flex ítems pueden romper la línea del eje horizontal si es necesario, para así conservar las características de las dimensiones de los flex ítems. Es de izquierda a derecha y de arriba abajo. Mueve hacia abajo la última caja. La mayoría de las veces vamos a querer que este en wrap.
  + **:wrap-reverse;** idem wrap, pero se ordena de izquierda a derecha y de abajo a arriba. Mueve hacia arriba la última caja.
* **justify-content:…;** Permite alinear los elementos en el main axis de la línea actual del flex container. Reagrupa los elementos de forma horizontal. Actúa cuando los flex ítems no utilizan todo el espacio disponible en su main axis. Es como se debe distribuir el espacio disponible entre los flex ítems.
  + **:flex-start;** Es el valor por defecto. Alinea los flex ítems en el main start del flex container (alineados desde la izquierda)
  + :**flex-end;** Alinea los flex items en el main end del flex container (alineados hacia el lado derecho).
  + :**center;** Alinea los flex ítems en el centro del flex container.
  + **:space-between;** los flex ítems tomarán la misma distancia de espaciado entre ellos dentro del flex container , quedando el primer y último elemento alineados con los bordes del contenedor en el eje principal.
  + **:space-around;** similar al space-between, solo que se espacian también el primer y último item de los bordes.
* **align-items:…;** Nos permite establecer la alineación que tendrán por defecto los flex ítems en la dirección del cross axis. Reagrupa los elementos de forma vertical.
  + **:flex-start;** valor por defecto. los ítems se apilan en el cross start del flex container.
  + **:stretch;** Valor por defecto. Trata de llenar toda la altura desde el cross start hacia el cross end del flex container, siempre y cuando los ítems no tengan propiedades de dimensión definidas.
  + **:flex-end;** los ítems se apilan en el cross end del flex container.
  + **:center;** los flex ítems se apilan en el centro del cross axis del flex container.
  + **:baseline;** alinea los flex ítems con su propia línea base. Utiliza como base la base del primer elemento.
* **align-content:…;** Alinea los flex ítems cuando estos no usan todo el espacio disponible en el cross axis del flex container, que de forma predeterminada es verticalmente. Esta propiedad solo tiene efecto cuando el flex container tiene varias líneas de flex ítems. Generalmente se lo usa para galerías.
  + :**stretch;** es el valor por defecto. Los flex ítems son mostrados con espacios distribuidos después de cada fila.
  + :**flex-start;** apila los flex items partiendo desde el cross start del flex container.
  + :**flex-end;** apila los flex items hacia el cross end del flex container.
  + :**center;** ajusta los ítems hacia el centro del flex container.
  + :**space-between;** los flex ítems se toman la misma distancia de espaciado entre ellos dentro del flex container.
  + **:space-around;** muestra los flex ítems con el mismo espacio de separación entre sí, solo que se espacian también del primer y último borde.

#### Propiedades para los flex ítems (hijos)

Son aquellas propiedades que se le pueden asignar a los ítems. Algunas de ellas sobrescriben propiedades que se configuran en el container. La única forma de que apliquen estas propiedades, es que ya haya un padre que declare el flex.

* **flex-grow:…;** Define la capacidad de un elemento de crecer cuando en el contenedor todavía hay espacio sobrante. Esta propiedad se configura como un valor numérico positivo. Por defecto viene configurado en 0 (no crece de manera horizontal). Si se lo configura en 1 para todos los ítems, todos ellos crecerán de igual manera, y si a alguno de ellos le configuro en 2, dicho elemento ocupará el doble que los demás.
  + Se le dice al hijo que ocupe más espacio que el hermano.
  + Para alinear una barra de navegación, es necesario utilizar el flex-grow en el css que haga referencia a la barra.
* **flex-shrink:…;** Configura la posibilidad de un elemento de encogerse en caso de que el máximo ancho del contenedor sea alcanzado por todos los elementos.
* **align-self:…;** Es para alinear un solo elemento respecto del cross axis.
  + **:auto;**
  + **:flex-start;**
  + **:flex-end;**
  + **:center;**
  + **:baseline;**
  + **:stretch;**
* **flex-basis:…;** Es similar al flex-grow pero con porcentajes. La suma de todos los flex-basis de los diferentes hermanos debería dar 100%. No se lo suele utilizar mucho.

#### Primera entrega del proyecto final

Entregar 5 archivos html con css, todos con colores, modificación de textos, box modeling.

Todos los headers y footers deben ser iguales.

Todos deben tener contenidos. Se debe estar la distribución del contenido aplicada.

Los colores (recomendable 3, máximo 5) se deben repetir en toda la página.

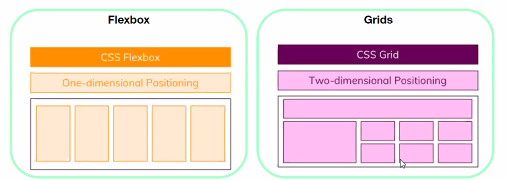
No es necesario utilizar flexbox en esta primera entrega.

# Clase 6: GRIDS

Grid es un layout que permite alinear elementos en columnas y filas. Permite la maquetación de contenidos en 2d (a diferencia de flexbox que es unidimensional).

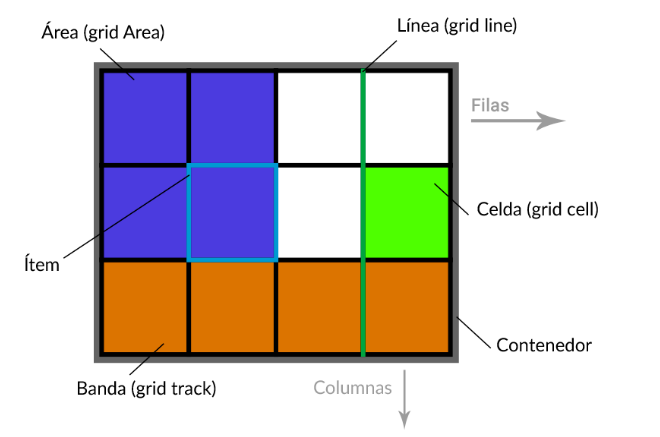
Se logra con grids una maquetación más dinámica y menos lineal.

Con grids se pueden implementar espacios vacíos.



Siempre hay que contar la cantidad de filas y las columnas.

#### Conceptos básicos



* Contenedor (grid container): Es el elemento padre que definirá el grid (cuadrícula). Es el elemento que contiene el **display:grid**.
* Item (grid item): Son los elementos que son hijos directos del contenedor.
* Celda (grid cell): Es la unidad mínima del grid.
* Línea (grid line): Separador horizontal o vertical de las celdas del grid y lo separa en filas y columnas.
* Área (grid área): Espacio rectangular delimitado por cuatro grid lines. Puede tener cualquier cantidad de celdas.
* Banda (grid track): Es el espacio entre dos grid lines, pudiendo ser horizontal o vertical.

#### Filas y Columnas explícitas

Es posible crear grids con un tamaño definido. Para ello, se utilizan las siguientes propiedades que sirven para indicar las dimensiones de cada grid cell, diferenciando entre filas y columnas:

* **grid-template-columns: …;** Establece el tamaño de las columnas (eje horizontal).
* **grid-template-rows:…;** Establece el tamaño de las filas (eje vertical).

Hay que aclarar (separadas por espacios e incluyendo la unidad de medida) el tamaño que tendrá cada columna y cada fila. Cada número representa una fila/columna.

La unidad fr es una unidad utilizada solo para grids que significa fraction. Representa la fracción del espacio disponible en el grid container. Si tengo grid-template-columns: 2fr 1fr, significará que tengo dos columnas donde la primera mide el doble que la segunda.

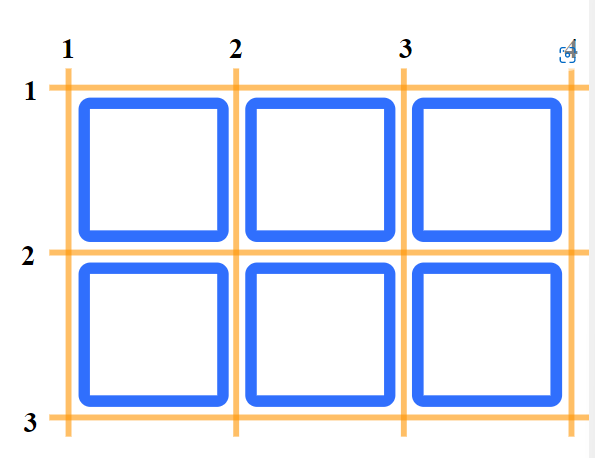
Siempre a los tamaños de los grids hay que ponerlos en fr, es una mala práctica ponerlos en px.

#### Filas y columnas repetitivas

En las propiedades grid-template-columns y grid-template-rows podemos indicar expresiones de repetición, indicando celdas que repiten un mismo patrón de celdas varias veces. repeat([núm de veces], [valor o valores])

#### Posicionar elementos

Por defecto, los ítems dentro del grid se posicionan automáticamente con el orden que vienen. Si queremos posicionarlos de otra forma, podemos utilizar los números de grid lines. Estas son numeradas de izquierda a derecha y arriba abajo. Para un grid de 3x2 sería:



Para posicionar un elemento, tenemos primero que tenerle asignada una clase en nuestro html, y en el css le diremos en que filas y columnas empieza y termina con las propiedades de CSS **grid-row-start, grid-row-end, grid-column-start y grid-column-end**. Por ejemplo, tengo un item con la clase item1 y lo quiero posicionar:

.item1{

grid-row-start: 2;

grid-row-end: 3;

grid-column-start: 2:

grid-column-end: 3;

}

Y de forma abreviada se puede escribir como:

.item1{

grid-row: 2/3;

grid-column: 2/3;

}

que decirle en que filas y columnas empieza y termina.

#### Espacios entre filas y columnas

Por defecto, los grids tienen todas sus celdas una al lado de la otra. Se puede dar un margen utilizando:

* **row-gap:…;**
* **column-gap:…;**

#### Grids por áreas

Es otra manera de posicionar ítems dentro del grid, utilizando grid áreas nombradas con **grid-template-areas** donde se deberán especificar el orden de las áreas, y luego en cada item hijo se debe utilizar **grid-area** para indicar el nombre del área que se trata. Cada apertura de comillas cuenta como una fila (row) y cada palabra dentro de las comillas contará como una columna. También se pueden utilizar palabras claves especiales: la palabra clave none indicará que no se coloca ninguna celda en esa posición, mientras que uno o más puntos indicará que se coloca una celda vacía en esa posición.

Ejemplo:

.grid {

display:grid;

grid-template-areas: "head head" "menu main" "foot foot";

}

.a { grid-area:head; background:blue }

.b { grid-area:menu; background:red }

.c { grid-area:main; background:green }

.d { grid-area:foot; background:orange }

En este ejemplo se declara un grid de 3 filas y 2 columnas, en donde el elemento de clase a (head) ocupará toda la parte superior, el elemento de clase b (menú) ocupará el área izquierda de la cuadrícula debajo del head, el elemento de clase c (main) ocupará el área derecha de la cuadricula debajo del head y el elemento de clase d (footer) ocupará la zona inferior del grid.

#### Posición de los elementos

Se pueden utilizar propiedades para colocar ítems dentro de la cuadricula desde el padre (grid container):

* **justify-items:…;** Distribuye los elementos en el eje horizontal.
* **align-items:…;** Distrinuye los elementos en el eje vertical.

También existen propiedades para cambiar la distribución de todo el contenido en su conjunto:

* **justify-content:…;**
* **align-content:…;**

#### Propiedades de los ítems

Las propiedades antes descriptas son para el contenedor padre. Las siguientes aplican a los ítems hijos del grid, para alterar o cambiar el comportamiento específico de dicho elemento.

* **justify-self:…;**
* **align-self:…;**
* **grid-area:…;**
* **grid-column: …/…;**
* **grid-row:…/…;**

#### Propiedades del padre

* **display: grid;** Es la manera de declarar un grid.
  + **:grid;** El grid aparecerá como contenido en bloque (encima/debajo) del contenido exterior.
  + **:inline-grid;** Permite que el grid aparezca como contenido en línea (izquierda/derecha) del contenido exterior.
* **grid-template-areas:…;** Es para asignar que espacio ocupará cada elemento. Dentro del argumento, cada apertura de comillas contará como una fila (row), y cada palabra dentro de las comillas contará como una columna. Si quiero dejar un espacio vacío, debo utilizar la palabra none.
  + Cada uno de los nombres que utilicé en el grid lo tengo que convertir en una clase, y a esa clase debo asignarle la propiedad **grid-area:…;** donde en los puntos suspensivos debo ponerle el nombre de la clase que creé.
  + A esa clase, se la debo asignar a los elementos html que quiero.
* **grid-area:…;**
* **grid-template-columns:…fr;** fr es una unidad de medida que se utiliza solo en grids. Van tantos números con fr como tantas columnas tenga.
* **grid-**template-rows:…fr;
* **column-gap:…;** Es el espacio de separación entre columnas.
* **row-gap:…;**

Para hacer galerías de imágenes, se suelen hacer con background-image. Para ello, en html creo divs vacíos con la clase usada en el grid. Si se hace esto, es importante definir las dimensiones (height y width) para que el contenedor (el grid) tenga tamaño. Primero se define el tamaño del contenedor y luego se lo fracciona.

Propiedad de background:

* **background-image:…;**
* **background-position:…;**
* **background-size:…;**
  + cover
  + contain: Se reduce de manera proporcional
* **background-repeat:…;**

Volviendo con grids, el máximo de columnas que se puede tener es 12.

LOS GRIDS SIEMPRE SE MIDEN POR LOS BORDES.

# Clase 7: GRIDS II

#### Grid mobile first

Son aquellos sitios/apps que son pensados primeros para desarrollarse para celulares. Se piensa primero de esta manera, y luego se ve el diseño para escritorio. Se parte del diseño del celular, y luego el resto de los dispositivos.

Generalmente la distribución de contenidos se puede plantear mejor si se hace mobile first. La distribución de contenidos tiende a ser más fácil de esta manera.

#### Diseño responsive

Es la idea de que un sitio web debe verse igual de bien en todo tipo de dispositivo, independientemente el tamaño de la pantalla.

Primero se diseña el sitio web desktop, y luego se lo para para móviles.

#### Media Queries

Es la base del diseño responsive, tanto para mobile first como para diseño responsive propiamente dicho. Es la manera de aplicar condicionales según el tamaño del dispositivo. Es como se va a mostrar nuestro sitio según el tamaño de la pantalla. Las media queries le dicen al navegador que reglas debe ignorar o aplicar según el dispositivo del usuario. Nos permiten presentar el mismo contenido HTML con diseños basados en CSS distintos.

Los media queries siempre comienzan con @media seguida de algún tipo de instrucción condicional, y luego llaves. Dentro de estas se insertan las reglas de CSS comunes, y el navegador prestará atención a estas reglas solo si se cumple la condición.

Se colocan al final del código CSS. Se comienzan como:

* **@media screen() and () {**

**}**

Media es un condicional. Nos dice en screen el tamaño mínimo de pantalla y en and el tamaño máximo de pantalla. Ej: para Tablet sería @media screen (min-width: 576px) and (max-width: 992px). Las media queries siempre se escriben en pixeles. Lo que nos dice el ejemplo es que lo que escribamos en el media quieries se verá solo en pantallas que entren en el rango del tamaño mínimo y el máximo de pantalla.

Media queries para celulares:

* + @media screen and (min-width: 320px) and (max-width: 575px){}
    - En el 99% de las veces siempre debe haber una sola columna. En el 1% restante son galerías cuadriculadas que pueden tener hasta 3 columnas.
    - Las navegaciones en mobile se ponen verticales. Es decir, se tendrían que poner con un flex-direction: column; Hay que ponerle un height al header, (height:auto) para que se pueda ver correctamente.

Media queries para tablets:

* + @media screen and (min-width: 576px) and (max-width: 992px){}

Para escritorio, no es necesario declarar media queries ya que es lo que venimos programando nosotros.

Dentro de las media queries hay que poner las cosas que cambiarán si el tamaño del dispositivo está en el rango marcado.

Para hacer responsive los grids, si o si hay que definir los media queries.

Para modificar alguna clase por ejemplo, debo copiar toda la clase dentro del media que quiero modificar, y solo debo agregar las propiedades que yo quiero cambiar.

#### Break Points

Son aquellos segmentos donde nosotros dividimos nuestros dispositivos.   
Nosotros vamos a utilizar los break points de boostrap.

Nosotros vamos a hacer las media queries de small (celulares), médium (tablets) y large (escritorio).

# Clase 8: Animaciones, transformaciones y transiciones

Se colocan generalmente en botones y en algún que otro texto a destacar. No está bueno llenarla de animaciones a toda la pagina web.

#### Gradientes

Se definen alrededor de un eje (x, y, z).

Permiten hacer fondos en degradé. Se pueden utilizar para agregarle una especie de filtro a las imágenes.

La manera de escribirlo es utilizando en **background-image:**

* **linear-gradient (to direccion, color1, color2)**
  + Dirección puede ser **top, right, left** o **bottom.**
* **radial-gradient.**

Recomendaciones:

* Al hacer gradientes, no usar más de tres colores.
* No aplicar gradientes a tarjetas/contenido con texto.
* Siempre debe leerse el texto, sin importar el color.
* Usar gradientes de colores similares.

Se pueden usar gradientes para textos. Para ello…

Se pueden usar gradientes para agregarles filtros a las imágenes. Para ello, debo utilizar gradientes con transparencia y utilizar la pseudoclase :before.

Los gradientes se suelen utilizar en el header/footer, en botones. En botones queda bien ponerle un gradient en la clase del botón, y en la pseudoclase :hover (este es para que cambie el estilo cuando se pasa el mouse por arriba)

#### Z-index

Es una propiedad de css para ordenar los elementos. Por defecto, todos los selectores de CSS tienen un z-index de 0, pero si le pongo un valor mayor, se superpondrá por encima de los demás.

#### Transformaciones

Nos permite cambiar como se ve un elemento con respecto a otro o a si mismo. Generan un ligero cambio. Generalmente se utilizan sobre los :hover. La idea es que las transformaciones sean lo más sutiles posible.

Para usarlos, debo utilizar la propiedad de CSS:

* **transform:…;**
  + **:translate(dist x, dist y);**
  + **:scale(ancho** [eje x]**,alto** [eje y]**);**
  + **:rotate(grados a rotar);**
    - Valor positivo: rota hacia la derecha.
    - Valor negativo: rota hacia la izquierda.
  + **:skew (def en grados eje x, def en grados eje y)** Es para deformar objetos.

#### Transiciones

Es para que los cambios de las transformaciones sean más sutiles. Todas las transformaciones deberían ir acompañadas por transiciones.

Para utilizarlas, debo utilizar la propiedad de CSS:

* **transition: elementos\_a\_usar tiempo;**
  + Los elementos a usar puede ser all, width, height.
  + tiempo recomendado: corto, ej 1s.

#### Animaciones

Sirven para hacer loops, donde el elemento se animará desde la carga de la web. Es la unión de dos partes: por un lado, una línea de tiempo (llamada keyframe) con la información de los cambios, y por otro, aplicar ese keyframe a un elemento que será el animado.

Las animaciones que se suelen usar son de librerías, se pueden ver algunas desde <https://animate.style/>.

# Clase 9: GIT

Es un controlador de versiones, está diseñado para manejar pequeños y grandes proyectos de manera eficaz.

Un controlador de versiones permite salvar nuestro proyecto en ciertos puntos dependiendo de los cambios que hagamos. En caso de que se haya cometido un error, se puede volver a una versión anterior. Registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.

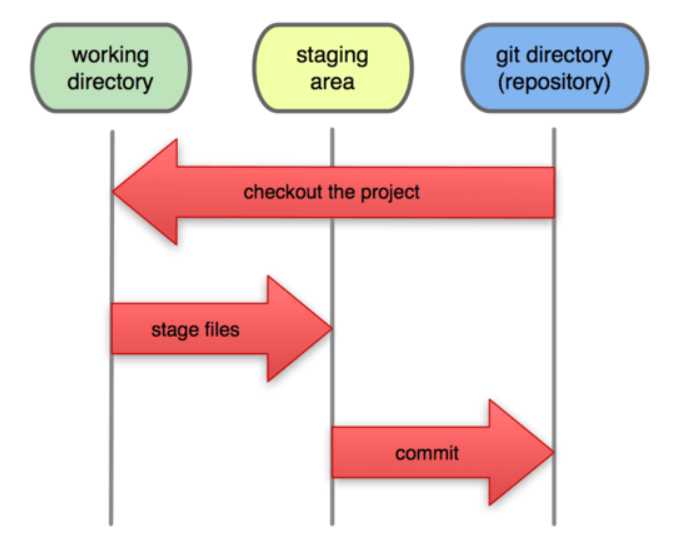
Este sistema registra los cambios de archivos en todas las etapas del proyecto.

En GIT, se puede ir y venir en las diferentes etapas del proyecto. Es muy útil para corregir errores.

#### Los tres estados de GIT

Git tiene tres secciones principales de trabajo:

* Working directory (directorio de trabajo). Es en el que trabajamos, “preparamos las cajas”.
* Staging areas. Aca se agregan los archivos que van quedando listos “agregamos las cajas listas”.
* repository. Registro de todos los archivos. “Lote listo”.



El flujo de trabajo en git es: creo los archivos en la carpeta del proyecto (working directory), luego indico cuáles voy a enviar al repositorio (stagging área) y finalmente los envío al repositorio principal (repository).

GIT siempre se usa de manera local (en nuestra computadora).

#### Comandos básicos de la terminal

* **cd nombre­­\_carpeta**: nos permite entrar a una carpeta.
* **cd**: vuelve una carpeta atrás.
* **dir**: Me permite ver en que carpeta estoy parado, y las carpetas que tengo para escoger.
* **clear**: limpia la consola.

Llego hasta la carpeta, y tiro el comando una única vez por proyecto:

* **git init nombre\_repo:** Inicio el git en nuestro proyecto. Con esto creamos el repositorio (de manera local). Se hace solo una vez por proyecto.

Un repositorio es el espacio donde nosotros centralizamos la información, donde nosotros pondremos nuestro proyecto.

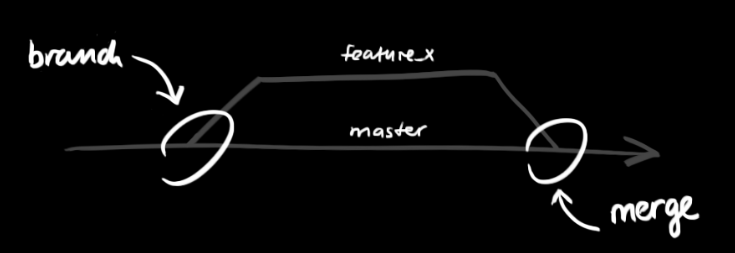
* g**it status:** nos dice el estado de nuestro repositorio local (working directory). Lo que aparece en rojo, nos quiere decir que los archivos todavía no los metimos en la caja, es lo que se ha modificado y hay que pasarlo al stagging área.
* **git add nombre­\_archivo:** Nos permite “meter las cosas a la caja”, es decir, agregar los archivos al stagging área. Podemos meter archivo por archivo. Si escribo **git add .**, introduciré a la caja todos los archivos.
  + Lo que nos sale en verde son los archivos que ya están en la caja, que están listos para hacer un commit en nuestro repositorio.
* **git reset nombre\_archivo:** Elimina un archivo del add, lo saca de la caja. Si uso **git reset** solo, quito todos los archivos del add.
* **git commit -m “nombre\_del\_commit”**: Una vez que nuestros archivos están en el stagging área, debemos pasarlos a nuestro repositorio local con este comando. Cierro la caja y la dejo lista. Con el **-m** y el siguiente texto entre comillas, dejo un mensaje del commit que armé. El mensaje debería de ser descriptivo de la entrega que armé. Commit es el comentario del cambio que hicimos.
* **git log:** Me muestra los commits que hice. Para una versión corta, puedo escribir **git log –oneline**.
* **git checkout hash­\_versión:** Nos permite Volver a una versión anterior del proyecto. No se la recomienda hacer a menos de que sea extremadamente necesario.

#### Ramas

Permiten hacer cambios sin afectar al resto de nuestros compañeros. Las ramas (branch) son caminos que puede tomar cualquier desarrollador dentro del proyecto.

La rama principal de los proyectos es la “Master”, y se pueden crear nuevas ramas para que otros desarrolladores del equipo trabajen sobre el proyecto en nuevas funcionalidades.

Llegado el momento, ocurrirá que aquellos cambios hechos en las ramas se quieran pasar a producción, entonces se hará un proceso de fusión entre la rama experimental y la rama original, operación conocida como merge.



Siempre se debe hacer un commit en las ramas en donde estoy trabajando, nunca un commit en master.

Las ramas las crea uno mismo, y se le pone el nombre de lo que se quiera desarrollar.

Comandos para trabajar con ramas:

* **git branch nombre­\_rama:** creo una rama con el nombre que le dé. Esa rama es una copia actual de la master, una réplica.
* **git branch -l**: lista las ramas que tenemos disponibles.
* **git branch -D rama\_a\_eliminar:** Elimina una rama.
* **git branch -m nombre\_rama nuevo\_nombre\_rama:** Renombra una rama.
* **git checkout nombre\_rama:** Me cambio de rama a la indicada para así poder trabajar sobre ella.
* **git merge rama\_a\_unir:** Es la versión que se genera cuando uno una rama a la master. Es la acción que se utiliza para combinar dos ramas de trabajo. Para hacer un merge, debo ir primero con **git checkout** a master, y luego le tiro el merge con la rama que quiero unir.
* **git pull master**: baja todos los cambios de master a la rama de donde estoy trabajando.

# Clase 10: Github

Github es una red social para compartir código. Es un servidor donde se sube nuestro código, nuestro repositorio, y se puede trabajar en equipo.

Se crea la cuenta de Github y se crea el repositorio…

Voy a la configuración del repositorio, a pages, source: escogo main, y le doy a save. Con eso obtengo un enlace de la página que subo a github.

Creo un nuevo archivo llamado .gitignore, a la misma altura del index de mi proyecto, y dentro de el creo una sola línea teniendo **node\_modules/**

Luego, agrego con **git add .** y **git commit -m “”**, y finalmente utilizo un nuevo comando: **git push** para subir a github nuestro proyecto.

.gitignore sirve para decir lo que quiero que se ignore al subir a github. Ahí le pido que se ignore que se suba una futura carpeta llamada node\_modules/, la cual es muy pesada.

# Clase 11: Frameworks CSS + Bootstrap

Librerías: Colección de elementos/rutinas/código que están juntas todas en un solo lugar, que suelen tener un propósito completo. Es el conjunto de instrucciones.

Frameworks: Toman las librerías e implementan una metodología o marco de trabajo. Le agrega a las librerías la metodología para trabajar. Es como si fuera un conjunto de reglas, mientras que las funcionalidades se las dan las librerías.

#### Boostrap

Es un framework con la particularidad de tener todo pensado para solucionar el problema del responsive.

Se importa Boostrap via CDN. Para ello, se importan los dos links que salen en la pag de boostrap. El primero se pone por encima de nuestro CSS.

El segundo link se pega en la última línea del body (antes de que este se cierre).

Le agrego los dos links a todas las páginas.

Utilizar Boostrap implica copiar el código de su documentación, y luego personalizarlo en función de lo que yo quiera.

Boostrap trabaja sobre un sistema de grids de 12 columnas.

Para hacer responsive, se utilizan los tamaños de pantalla (breakpoints). Los que más se usan son…

.

.

.

Clase img-fluid: hace que el tamaño de las imágenes se adapte al tamaño del contenedor.

Recomendaciones de uso:

* navegación,
* formularios,
* carrousel.

## Video curso Bootstrap

#### Container

Son el contenedor fundamental donde estará todo dentro. Para utilizarlos, hay que usar la clase container, pero también puede ser container-{breakpoint} o container-fluid.

#### Grids

Para trabajar con grids, todo debe ser envuelto con un elemento clase container. Luego, dentro del elemento debo generar un elemento **row**, y dentro de este debo generar los elementos **col**.

* Importante, toda **row** debe tener por lo menos un elemento **col** dentro.
* Si la clase que utilizo es solamente col, todas las columnas dentro del **row** ocuparan exactamente el mismo tamaño.
* Si quiero utilizar columnas con tamaño, tengo que usar la clase **col-{num}**, donde en **num** tiene que ir un número del 1 al 12. Esto es porque en Bootstrap el sistema de grids que utiliza es de 12 columnas. El número total entre todas las columnas de una misma fila debe ser igual a 12.
  + Si me paso de 12 en la suma de columnas dentro de una fila, automáticamente la última pasará a una nueva fila.
* Si se quiere cambiar el sistema de grids según el tamaño de la pantalla, debo utilizar los breakpoints. Para ello, debo poner **col-{br}-{num}**, donde **br** es el breakpoint que vamos a utilizar. Esto nos hará que para las resoluciones menores a las del breakpoint, se vea solo una sola columna, y para las resoluciones mayores veremos la cantidad de columnas que le definimos con el num.
  + Se puede aplicar varios breakpoints en una misma columna, utilizando varias clases **col-{br}-{num}** con distinto br y distinto num, para que cambie el comportamiento esa columna según el tamaño de mi pantalla.
* Se pueden anidar columnas dentro de filas.

Alineación vertical (de filas y columnas)

* Utilizo la clase **align-items-{posición}**, en donde posición puedo poner **start, end, center.** Esta clase la utilizo en las row, de modo que alineo todas las columnas por igual.
* Utilizo la clase **align-self-{posición}**. Esta clase la utilizo en cada columna, de modo que alineo cada col por separado.

Alineación horizontal (de filas y columnas)

* Utilizo la clase **justify-content-{posición}**, en donde posición puedo poner **start, center, end, around, between, evenly.**

Offsets

Son huecos entre columnas. Lo hago con la clase **offset-{numero}** donde en número coloco el número de columnas que quiero utilizar de espacio (de izquierda a derecha)

Gutters

Son los espacios entre las diferentes columnas.

Se dan con las clases **gx-\*** para controlar el espaciado horizontal entre columnas, donde en **\*** va el valor del tamaño, que es un número de 1 a 5. Se le aplican al elemento con clase row.

También se pueden utilizar las clases **gy-\*** para controlar el espaciado vertical entre filas (se le aplican al elemento con clase row).

Utilizando las clases **g-\***, controlo el espaciado horizontal y vertical entre columnas.

# Clase 12: @media + pseudo clases

#### Pseudo clases

Se reconocen porque empiezan con dos puntos. Algunos ejemplos son:

* :hover
* :visited. Estilo que se le aplica a un elemento que ya ha sido visitado.
* :not(elemento). Es para que al elemento nombrado no se le aplica el estilo definido.
* :nth-child(numero). Le aplica las propiedades al número de hijo del elemento.
* :first-child. Afecta solo al primer elemento.
* :focus. Se suele usar en los inputs. Hace que cambie el estilo cuando se está haciendo foco sobre el elemento.

#### Pseudo elementos

Agregan elementos a nuestro código. Se usan mucho para hacer arte en css.  
Se identifican porque tienen dos dos puntos. Incorporan una especie de contenedor que se incorpora a la clase que yo le estoy diciendo.

* ::after
* ::before

# Clase 13: SASS I

#### Procesacores de CSS

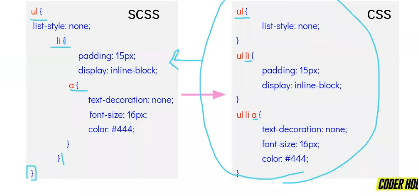
SASS es un preprocesador de css. Es mas sencillo de mantener, es mas ordenado, nos permite tener variables, nod permite optimizar el código.

Preprocesado es código que se genera automáticamente.

Se pueden crear funciones, tener variables, usar condicionales.

La sintaxis que va mos a usar es sassy CSS. Es similar a la sintaxis nativa de CSS.

SASS nos va a permitir hacer la anidacion (nesting) de clases. Se va a tener clases padre y clases hijo (en la metodología BEM, el bloque sería la clase padre y el bloque\_\_elemento la clase hijo).



Voy a tener que reescribir todas las clases como el tipo de SCSS.

Componentes: todo aquello que ocupe más de una página (ej: footer).

Por cada componente podemos crear una página scss. Todas las hojas de estilo que no sean la principal deben llevar antes del nombre de la clase un guion bajo \_. Creo una nueva hoja scss para por ejemplo el footer, header, mediaqueries, navegación, etc.

Luego, por cada segmentación debo importarlas a la principal con @import “componente”; en donde en componente no tengo que poner el \_ ni la extensión del archivo. Es normal que las mediaqueries se coloquen al final del archivo scss.

Es común que el archivo principal sea puros @import.

De ahora en adelante, cada vez que vaya a trabajar en el archivo SASS debo utilizar en la terminal el comando npm run watch-css, y para salir del npm tengo que apretar ctrl c.

En SASS, se pueden también crear variables. Las variables siempre se crean en una hoja de estilos diferente, y se la importa primero que todos los demás elementos.

Las variables nos permitirán reutilizar la información que se repite. Las variables más básicas a guardar son tamaños, letras y colores.

Todas las variables tendrán la siguiente estructura: **$nombre-variable: contenido;**. En el contenido puede ir por ejemplo, un color.

Luego, para utilizar la variable en los archivos SCSS debo llamarla al igual que como la había nombrado, incluyendo el $.