

Web Developer Fundamentals

Bienvenida

Veremos 2 estándares de los 3 que se usan en el desarrollo web: HTML y CSS.

Perfiles de un web developer

Frontend

Es el desarrollador que maneja la parte visual, la parte del cliente (navegador)

- Interacciones
- Animaciones
- Estilos
- Navegación

Estándares que maneja un front

- HTML
- CSS
- JavaScript

Frameworks CSS

Los frameworks nos ayudan a la hora de construir nuestro proyecto. Son ciertos fragmentos de código para acortar la creación de un producto. Algunos son:

- Bootstrap
- Tailwind CSS
- Foundation

Frameworks y librerías JS

Son productos que nos permiten escalar nuestro proyecto a producción de una forma más rápida y con una mayor interacción. Algunos son:

- React
- Angular
- Vue

Preprocesadores CSS

Es una forma diferente de hacer CSS. Algunos lo denominan que es usar "css con superpoderes".

- Less
- Stylus
- Sass

Compilador/Empaquetador

Estos programas nos permiten usar las últimas versiones de JavaScript (Siempre debemos usar las últimas versiones de JS, es muy buena práctica) mediante un proceso de compilado para que el navegador lo entienda.

- Babel
- Webpack

Backend

El desarrollador backend es aquel que trabaja del lado del servidor. Maneja toda la lógica sobre cómo encontrar a los usuarios y demás funciones. Un back no suele necesitar estándares, este puede manejar diferentes tecnologías y lenguajes de programación.

Lenguajes de programación

- Python
- Node.js
- PHP
- Ruby
- Go
- Java
- .Net

Frameworks

- Django para Python
- Laravel para PHP
- Rails para Ruby
- Express para Node.js
- Spring para Java

Infraestructura

La infraestructura o nube es usada para temas de deploy. Esto puede que no haga parte del backend, sin embargo es necesario tener conocimientos en ellas. De aquí puede salir un perfil llamado DevOps.

- Google Cloud
- Digital Ocean

- AWS
- Heroku

Bases de datos

- MongoDB
- MySQL
- PostgreSQL

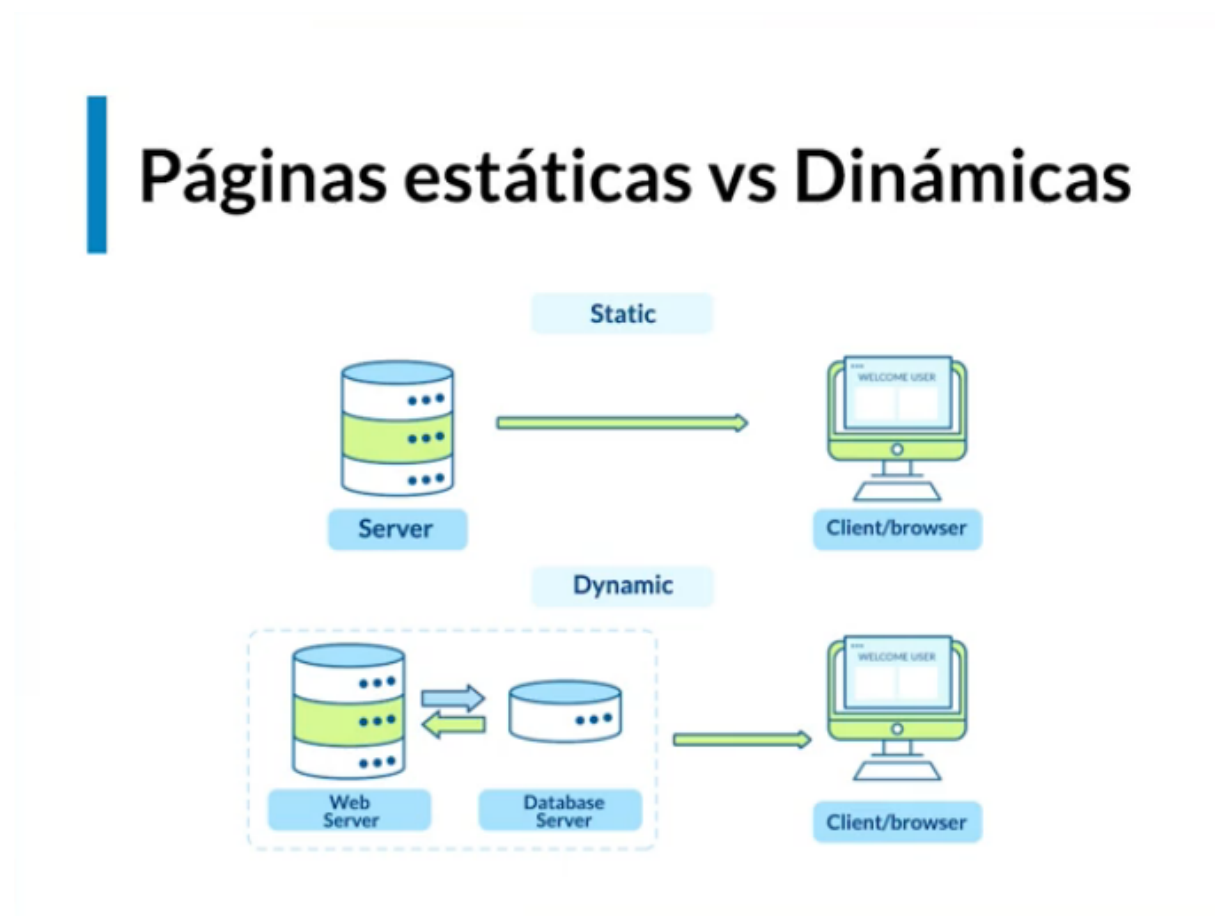
FullStack

Es un desarrollador que fusiona varias herramientas/conocimientos del frontend y el backend. Las librerías, los frameworks y lenguajes de programación. Un full stack no lo sabe todo 100%.

Un full stack, mejor dicho, es un desarrollador que entiende todo el proceso del desarrollo, desde el diseño y la idea, hasta el código y el despliegue. ***Es entenderlo todo, no saberlo todo.***

Escuela de desarrollo web → Full Stack. Pero con especialización Front o Back.

Páginas estáticas vs dinámicas



Los websites estáticos son aquellos que por lo general no cambian. Se les conoce también como landing page o page information.

- No están conectadas a una base de datos
- Por lo general no están conectadas a un servidor propio

Por el contrario, los websites dinámicos son aquellos que dejan de ser páginas y se convierten más en aplicaciones por todas las cosas que se pueden ver y hacer.

- Si tienen bases de datos y servidor propio
- Tiene interacción
- Debe guardar registros e información

Siempre, dependiendo de los requerimientos del cliente, debemos decidir que tipo de página usar.

Estático: Blogs, landing.

Dinámicos: Twitter, Platzi. → Contenido que cambia.

HTML

Hypertext Markup Language

Anatomía de una página web

La anatomía de una página web se podría decir que son los elementos básicos de esta misma. Estos son:

- Header
 - Logo
 - Nav
- Main content
- Side Bar
- Footer

Index y su estructura básica: Head

En el head van todos los archivos importantes para el correcto funcionamiento de nuestra plataforma pero que no deben ser visibles por lo usuarios. Algunos de estos archivos pueden ser:

- Frameworks
- Librerías
- Estilos
- Fuentes

- APIs

```
<!DOCTYPE html>
```

```
<!--Le decimos al navegador que este archivo es del tipo html:5-->
```

```
<html lang="es">
```

```
<!--Es la etiqueta "padre" donde vivirá nuestro proyecto. El atributo lang establece el idioma del sitio web.
```

```
Debemos usarlo para que el navegador pueda traducir nuestra página-->
```

```
<head>
```

```
<meta charset="UTF-8" />
```

```
<!--Este atributo nos ayuda a la hora de incluir caracteres especiales y emojis en nuestro proyecto-->
```

```
<meta name="description"content="Esta página te mostrará fotos de gatos" />
```

```
<!--Muestra una descripción de nuestro sitio en los buscadores-->
```

```
<meta name="robots"content="index,follow" />
```

```
<!--Le dice a los robots de los navegadores que rastreen nuestra página y la muestran en las búsquedas-->
```

```
<title>Mi página</title>
```

```
<!--Título de nuestra página, no confundir con los H1-H6. Este titulo es el que ves en la pestaña del navegador-->
```

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

```
<!--Nos ayuda a trabajar en proyectos reponsive-->
```

```
<link rel="stylesheet"href="/css/style.css">
```

```
<!--Linkea/Enlaza archivos de estilos u otros archivos que necesitemos en nuestro proyecto-->
```

```
</head>
```

El usuario solo verá lo que está dentro del body.

*El primer archivo siempre debe llamarse *index.html* porque este es el primer archivo que buscan los servidores a la hora de cargar los sitios web.

*Existen etiquetas que se cierran y otras que no. Como por ejemplo:

```
<a href="platzi.com">Contenido</a>
```

```

```

Index y su estructura básica: body

Existen dos tipos de etiquetas en body: De contenido y contenedoras. Las de contenido son las encargadas de mostrar texto, imágenes, videos, etcétera... Mientras que las contenedoras son las que nos van a permitir darle estructura y organizar nuestro contenido.

<body>

<header><!--Sección superior de nuestro website-->

<nav></nav><!--Sección de navegación de nuestro website, siempre dentro del header-->

</header>

<main><!--Main es el contenido central de nuestro website, "la parte del medio"-->

<section>

<!--Nuestro website puede estar dividido por secciones, por ejemplo platzi tiene 3:
El navegador de cursos y rutas, el feed y nuestras rutas de aprendizaje-->

<article>

<!--Contenido independiente de la página. Es reutilizable-->

</article>

</section>

<!--Lista desordenada: Sin numerar-->

<!--Item List. Elementos de la lista-->

<!--Lista ordenada: Numerada-->

</main>

<footer><!--Sección final de nuestro website-->

</footer>

<p>Soy un texto</p><!--Párrafo, texto-->

<h1>Soy un titulo</h1>

<!--Títulos, muestran el texto más grande y con negrilla. Existen desde el h1 al h6-->

```
<a href="#">Soy un link</a>
```

```
<!--Enlaces/links que nos permitirán movernos entre páginas.-->
```

```
</body>
```

Anatomía de una etiqueta HTML







Etiquetas multimedia

Son el tipo de etiquetas que usaremos para insertar imágenes y vídeos.

Tipos de imágenes

Tabla de diferencias

	Categoría	Paleta	Uso
	Lossless	Máximo 256 colores	<ul style="list-style-type: none">• Animaciones simples• Gráficos con colores planos
	Lossless	Máximo 256 colores	<ul style="list-style-type: none">• Uso de transparencia• Sin animación• Ideal para íconos
	Lossless	Colores ilimitados	<ul style="list-style-type: none">• Similar a PNG-8• Maneja imágenes fijas de más colores y transparencia
	Lossy	Millones de colores	<ul style="list-style-type: none">• Imágenes fijas• Fotografía
	Vector / Lossless	Colores ilimitados	<ul style="list-style-type: none">• Gráficos / logotipos para web• Retina / pantallas de alta resolución

Existen 2 tipos de imágenes: **Con pérdida** (Lossy) y **sin pérdida** (Lossless) y esto dependerá del formato que manejemos.

- **Lossy**: Este tipo de imágenes se refiere a imágenes pequeñas y que pierden calidad (comparadas con las lossless). Nos ayudan a mejorar el tiempo de carga de nuestro website pero perdemos calidad. Algunos de estos formatos son:
 - JPG → Photographic Experts Group → Al comprimirla, pierde calidad.
- **LossLess**: Este tipo de imágenes no pierde calidad, su calidad siempre se mantendrá igual. Son más lentas en cargar y consume más recursos, pero tendremos unas mejores imagenes sin importar que se comprimen. Algunos formatos de estas son:
 - GIF → Graphics Interchange Format
 - PNG → Portable Network Graphics → PNG 8 equivale a 258 colores. → PNG 24 equivale a más de 256 colores.
 - SVG (Vector) → Scalable Vector Graphics → Se usan para pantallas de retina. Nunca pierden calidad. Nunca.

Optimización de imágenes

No es óptimo cargar imágenes en nuestro sitio web que pesen mucho porque tardan bastante tiempo en renderizarse y le dan una mala experiencia al usuario.

El tamaño promedio de una imagen debe ser de 70 a 100 kilobytes.

Podemos optimizar nuestras imágenes con los siguientes recursos:

[TinyPNG - Compress PNG images while preserving transparency](#)

Mejora y reduce el tamaño de las imágenes png y jpg.

[Ver Exif online . quitar Exif online](#)

Elimina los metadatos de la imagen. Se usa para imágenes que vienen de una cámara.

Etiqueta img, figure y figcaption

Es recomendable descargar imágenes solo de tamaño grande, mediano o pequeño.

```
<figure style="margin: 0;">
  
  <figcaption>Es una imagen de un perrito 🐶</figcaption>
</figure>
```

src → Donde está la imagen, su ruta. Puede ser desde nuestras carpetas o desde un enlace de la web.

alt → Texto alternativo por si no se renderiza la imagen y para mejorar la accesibilidad de nuestro sitio.

figure → Genera un contenedor para la imagen.

figcaption → Agrega una descripción a la imagen. Será visible en la parte inferior de esta misma. Va dentro de la etiqueta figura.

Por buenas prácticas siempre deberíamos usar la etiqueta figure para insertar un archivo multimedia.

Etiqueta video

Nos permite subir un video mediante la misma forma que una imagen, mediante el atributo src.

```
<section>
  <video controls preload="auto">
    <source src="/video.m4v#t=10,60" />
    <source src="/video.mp4#t=10,60" />
    <source src="/video.gif" />
  </video>
</section>
```

control → Aparecen los controles para manipular el vídeo. No recibe valores, es un parámetro vacío.

preload → Ayuda a que el video se empiece a descargar una vez se abre el navegador en esa página. A descargar, no a reproducirse automáticamente. Reproducirse automáticamente es una mala práctica.

source → Va dentro de la etiqueta video y se usa para especificar varias rutas en caso de que el navegador no entienda algún formato de vídeo. Se pueden poner varios formatos el navegador usa el que más le convenga. Para usarlo se elimina el atributo src del video y se le pone a las etiquetas source. Al video se le dejan los demás atributos.

Si queremos que el vídeo inicie y termine en un minuto/segundo específico, debemos usar unos atributos dentro del src:

- #t= → Indica el tiempo en el cual empezará y terminará.
- 10, 90 → Son los valores en segundos. Donde 10 (izquierda) es donde inicia y 90 (derecha) donde finaliza. Deben ir separados solo por una coma sin espacio.

Formularios

Etiqueta form e input

Los formularios son la forma en la cual podemos empezar a generar interacción con los usuarios. Debemos generar buenos formularios para los usuarios, deben ser cortos, fáciles de usar y directos.

<!-- Como no hacerlo -->

```
<div>
  <input type="text" />
  <input type="text" />
  <input type="text" />
  <input type="text" />
</div>
```

<!-- Como si hacerlo -->

```
<form action="">
  <label for="nombre">
    <span>¿Cual es tu nombre?</span>
    <input type="text" id="nombre" placeholder="John Cárdenas" />
  </label>
  <label for="inicio-platzi">
```

```

    <span>¿Que día comenzaste en Platzi?</span>
    <input type="date" id="inicio-platzi" />
</label>
<label for="horario">
    <span>¿En que horario estudias?</span>
    <input type="time" id="horario" />
</label>
</form>

```

form: nos permite decirle al navegador que lo que usaremos es un formulario y que el usuarios va a interactuar con el. Es una etiqueta contenedora.

action: Es la URL o base de datos a donde se va a enviar la información.

label: Es una etiqueta contenedora de **input**. Recibe un atributo importante llamado **for** y el cual debe ser el mismo que el **id** del **input**.

placeholder: Agrega información dentro del input. Sirve para indicarle al usuario un ejemplo sobre lo que debe hacer.

Tipos de inputs:

[HTML Input Types](#)

Calendar, auto complete y requiere

El atributo **name** de los inputs nos ayuda en el caso que queramos enviar la información a alguna URL. Debe ser igual que el atributo **for** de **label** y por lo tanto que el **id** del **input**.

```

<form action="">
  <label for="nombre">
    <span>¿Cuál es tu nombre?</span>
    <input
      type="text"
      name="nombre"
      id="nombre"
      autocomplete="name"
      required
    />
  </label>
  <label for="correo">
    <span>¿Cuál es tu correo?</span>
    <input
      type="email"
      name="correo"
      id="correo"
      autocomplete="email"
    >
  </label>
</form>

```

```

        required
    />
</label>
<label for="pais">
    <span>¿En que país vives?</span>
    <input
        type="text"
        name="pais"
        id="pais"
        autocomplete="country"
        required
    />
</label>
<label for="cp">
    <span>¿Cuál es tu código postal?</span>
    <input
        type="text"
        name="cp"
        id="cp"
        autocomplete="postal-code"
        required
    />
</label>
<input type="submit" />
</form>

```

autocomplete: Completa el input con los datos que tenga guardado el navegador del usuario. Recibe valores dependiendo de que busquemos autocompletar: *name, email, address***.*

require: Le indica al usuario que debe rellenar la información porque es requerida. Si no lo llena, no avanza.

Select

Select permite crear una lista para seleccionar varias opciones. Sirve de contenedor para la etiqueta **option**, la cual muestra los valores de la lista que crea **select**. Sin embargo, se debe usar más la opción **datalist** que **select**, ya que **datalist** junto a **input** con el **atributo list** le permite al usuario buscar en caso de que la lista sea muy larga.

```

        <select name="" id="">
            <option value="JavaScript">Curso de JavaScript</option>
            <option value="HTML5">Curso de HTML5</option>
            <option value="CSS3">Curso de CSS3</option>
            <option value="Web Standards">Curso de Web Standards</option>
        </select>

```

<!-- Con buenas prácticas -> Le das la opción al usuario de encontrar lo que está buscando -->

```
<input list="cursos" />
<datalist id="cursos">
  <option value="JavaScript"></option>
  <option value="HTML5"></option>
  <option value="CSS3"></option>
  <option value="Web Standards"></option>
</datalist>
```

Input type submit vs Button Tag

En HTML5 se pueden crear dos tipos de botones, con input y con button.

```
<body>
  <input type="submit" value="Nombre" />
  <button>Enviar</button>
</body>
```

Básicamente, deberíamos usarlos así:

input para formularios.

button para otro tipo de botones.

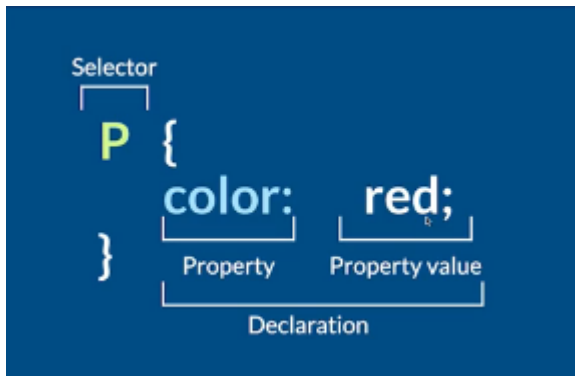
*El atributo **value** permite personalizar el texto de un input de tipo submit.

CSS

¿Que es?

Las hojas de estilo en cascada es el archivo que le aplica los estilos a un sitio web en forma de cascada, leyendo el código de arriba hacia abajo.

Anatomía de una regla CSS



¿Cómo usamos CSS? por selector, clase y/o IDs.

Para añadir CSS a nuestro archivo HTML existen 3 formas:

1. Link externo

```
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="./estilos.css" />
</head>
```

2. Se puede agregar como estilo embebido a una etiqueta HTML.

```
<a style="color: white">
```

3. Agregar los estilos dentro de la etiqueta **head**

```
<head>
<meta charset="UTF-8" />
<meta name="viewport" content="width=device-width, initial-scale=1.0" />
<link rel="stylesheet" href="./estilos.css" />
<title>Pseudo Clase y Pseudo Elementos</title>
<style>
  body {
    background-color: black;
  }
</style>
</head>
```

Para agregar estilos a una etiqueta de HTML usamos clases y IDs. Siendo las clases las más importantes para elementos generales y los IDs para elementos específicos (Son más recomendados para JavaScript) Ejemplo de usos de estilos con clases:

```
<nav>
```

```

<ul id="main-nav" class="nav">
  <li><a href="">Home</a></li>
  <li><a href="">Cursos</a></li>
  <li><a href="">Instructores</a></li>
  <li><a href="" class="blog">Blog</a></li>
</ul>
</nav>

```

```

.nav {
  margin-top: 10px;
  list-style: none;
  padding-left: 0;
}

```

```

.nav li {
  display: inline-block;
}

```

```

.nav a {
  color: white;
  background-color: #13a4a4;
  padding: 5px;
  border-radius: 2px;
  text-decoration: none;
}

```

```

.nav .blog {
  background-color: red;
}

```

Para este curso usaremos la metodología BEM la cual nos ayuda a generar clases a elementos de HTML de una manera más ordenada y específica. BEM usa esta sintaxis:

```

.main
.main-nav
.main-nav__item
.main-nav__item a

```

Con las clases e IDs de CSS siempre debemos ser lo más específicos que podamos.

Pseudo clases y pseudo elementos

Las pseudo-clases especifica un estado especial a un elemento seleccionado.

```

/* Ejemplo */
p:first-child {

```

```
color:red;
}
```

/*Convierte todo el texto del primero hijo del bloque de código en rojo. Todo el texto */

Lista de pseudo-clases: <https://developer.mozilla.org/es/docs/Web/CSS/Pseudo-classes>

A diferencia de las pseudo-clases los pseudo-elementos no describen un estado especial, sino que permiten añadir estilos a una parte concreta (a veces muy específica) de un elemento.

```
/* Ejemplo */
p::first-letter {
  color: red;
  font-size: 20px;
}
```

/* Toma solo la primera letra de un elemento, no todo su contenido. Solo la primera letra.*/

Lista de pseudo-elementos:
<https://developer.mozilla.org/es/docs/Web/CSS/Pseudoelementos>

:active → Pseudo clase que se activa cuando un elemento recibe el evento click.

::after → Pseudo elemento que agrega algo (texto, número, figuras) delante del elemento HTML especificado.

*En los proyectos casi siempre se trabaja una hoja de estilos por página. Por lo general el nombre del archivo de estilos es el mismo nombre de la página.

*propiedad **content** → Agrega contenido (texto) a un elemento especificado.

Diferencias

Pseudo classes	Define el estilo de <i>un estado</i> especial de un elemento.
:class	
Pseudo elementos	Define el estilo de <i>una parte</i> específica de un elemento.
::element	

Modelo de caja

En HTML siempre estamos trabajando con "cajas" que son contenedores el contenido que agregamos. Esta caja cuenta con 4 componentes:

- Margin → Espacio externo
- Border → Borde del contenido. Externo.
- Padding → Espacio interno
- Content (width y height) → Texto, imágenes, vídeos

**** → Es el selector universal de HTML. Se usa para resetear los estilos predeterminados por el navegador.

*Para hacer que un elemento tome el ancho que le asignemos más un margen o padding sin que se rompa o no funcione el estilo, podemos usar **calc**. Hará que el ancho se calcule tomando en cuenta el margen y ancho especificado. Esto es muy útil a la hora de hacer desarrollo responsive.

```
container {  
    width: (calc 50% - 20px)  
}
```

Herencia

Es el código CSS que le pasará al hijo de una etiqueta contenedora o padre. El valor **inherit** es el que nos permite heredar estilos de las **etiquetas padres**. Por ejemplo, si hay un texto contenido en una etiqueta div, se le puede asignar un **margin: inherit** y arrojará el mismo margen que tenga ese div.

Especificidad en selectores

La especificidad es muy importante ya que es el orden por el cual el navegador decide qué estilos aplicarle a un elemento. El orden en el cual se decide esto es el siguiente:

1. Importancia del selector
2. Especificidad
3. Orden de las fuentes → Como se mandan a llamar

Si 2 reglas tienen la misma importancia, pasa a evaluarse la especificidad de cada una. Si cuentan con la misma especificidad, pasa el orden a decidir cual se aplica.

En cuestión de importancia, estas son las reglas que primero se aplican:

1. !important
2. inline styles
3. #ids
4. .class
5. tag

Donde a **!important e inline styles** debemos evitarlos las mayorías de veces ya que son mala práctica. Siempre debemos tratar de usar las clases.

En cuanto al orden de las fuentes:

Si se mandan a llamar unos estilos arriba y otros abajo (los dos con la misma clase) los últimos llamados sobrescribirán a los primeros, porque tal como su nombre lo indica, los estilos CSS se leen en cascada y guarda los últimos cambios/valores que lee. (Este es un caso muy extremo)

Especificidad

Selectores	Especificidad
!important	1,0,0,0,0
Inline styles	0,1,0,0,0
#id	0,0,1,0,0
.class	0,0,0,1,0
tag	0,0,0,0,1



Calculadora de peso de especificidad:

[CSS Specificity Calculator. * CodeCaptain](#)

Juego para dominar los selectores:

[CSS Diner](#)

Demo de especificidad y orden de selectores

*Por cada etiqueta no se puede tener más de 1 ID. Pero cada etiqueta puede tener más de 1 clase, es normal. Aunque lo máximo recomendado son 3. En cambio el ID es único, no pueden haber 2 IDs en la misma página con el mismo nombre.

*Los IDs son más importantes para los estilos que las clases.

*VStudio nos muestra el nivel de especificidad de cada regla de CSS.

Combinadores: Adjacent Siblings (combinators)

Los combinadores nos permiten, como su nombre lo dice, combinar selectores y crear una mayor especificidad a la hora de establecer estilos. Los 4 tipos de combinadores más usados son:

```
/* 1. Hermano cercano: Aplica el estilo a todas las etiquetas p que tenga cerca un h2 */  
h2 + p {  
    color: red;  
}
```

```
/* 2. Hermano general: Aplica el estilo siempre y cuando coincida una p con un h2 en el  
mismo contenedor*/  
h2 ~ p {  
    color: red;  
}
```

```
/* 3. Hijo: Agregale estilos a una etiqueta p que sea hijo directo de div */  
div > p {  
    color: blue;  
}
```

```
/* 4. Descendiente: Todas las etiquetas de p que estén dentro de un div */  
div p {  
    color: red;  
}
```

Medidas

Las medidas en CSS es lo que nos permitirá establecer tamaños a fuentes, contenedores, iconos, imágenes, etcétera... Existen 2 tipos de medidas:

1. **Absolutas:** Es una medida que no cambia en ningún momento, se usan los pixeles.
2. **Relativas:** Son valores que sí cambian y son relativas a otros elementos.

Medidas EM

Em es un acrónimo de elemento y lo que hace es que toma el tamaño de fuente de su padre directo. Si un párrafo está encerrado en un div que tiene un tamaño fuente de 16px, entonces:

1em → 16px

2em → 32px

1.5em → 24px

No es de las mejores medidas porque puede generar conflicto entre medidas ya que siempre toma la de su padre más cercano y esto puede cambiar el tamaño de los elementos sin que nos demos cuenta.

Medidas REM

*La etiqueta HTML tiene 16px de fuente por defecto

REM hace referencia a la etiqueta **root** de nuestro archivo HTML, la cual es la etiqueta <html>. Por lo tanto 1rem siempre será igual a 16px a no ser que cambiemos el tamaño de font-size del html. REM puede ser confuso de calcular y para esto existe un truco, el cual consiste en establecer el tamaño de fuente del html en **62.5%** de esta forma:

1rem → 10px

1.6rem → 16px

2rem → 20px

Max/min Width

min-width: El elemento no se puede hacer menos ancho desde un tamaño especificado.

max-width: El elemento no se puede hacer más ancho desde un tamaño especificado.

min-height: El elemento no se puede hacer menos largo desde un tamaño especificado.

max-height: El elemento no se puede hacer menos largo desde un tamaño especificado.

```
section {  
  width: 80%;  
  min-width: 320px;  
  max-width: 500px;  
  min-height: 500px;  
  margin: 0 auto;  
  background-color: red;  
}
```

*Cuando usemos el min y max width debemos tener un width base y casi siempre será en porcentajes.

*En el caso de height no es necesario tener un height base. Se pueden establecer directamente los min y max.

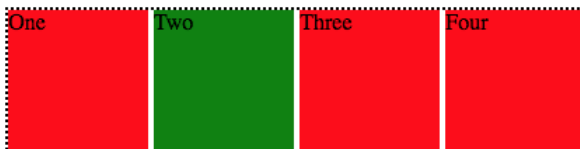
*Un problema de overflow quiere decir que se tiene más contenido que espacio en el contenedor padre.

Position

Es la forma en la cual podemos posicionar contenedores de CSS o las etiquetas de HTML. Algunas de las posiciones más usadas son:

1. **Static:** Es la que viene por defecto. El elemento se queda donde fue ubicado en el código y de ahí no se mueve.
2. **Absolute:** Esta posición le indica al elemento que se debe borrar del flujo normal de la página. Esto causará que el elemento quede por encima de los demás y deje de ocupar su espacio.
3. **Relative:** Esta posición es considerada una "normal" porque se ubica en con respecto a su posición original. Esto causará que el elemento siga ocupando su espacio a pesar de que no esté ubicado allí.

position: static



position: relative



position: absolute



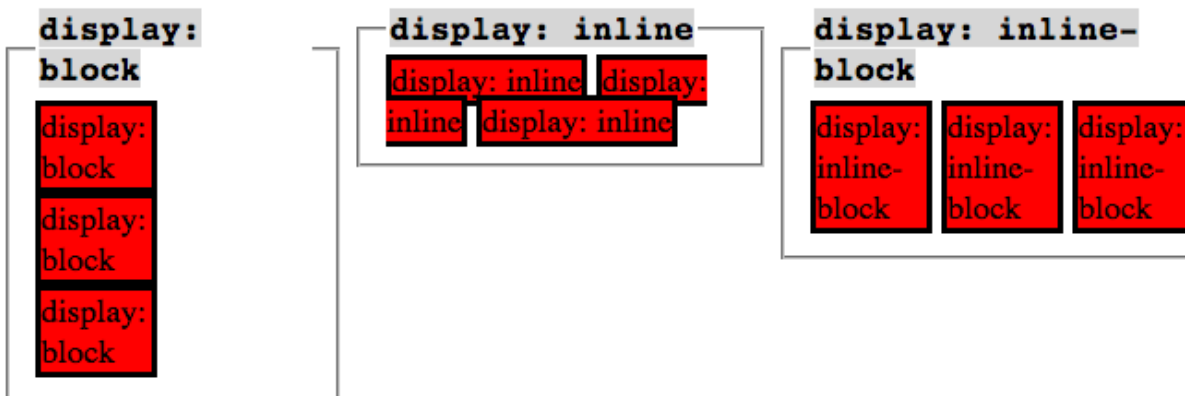
1. **Fixed:** Al igual que sticky, sirven para que el contenido se quede en su posición sin importar si se hace o no scroll. **Fixed mantiene el efecto desde que se abre la página.**
2. **Sticky:** Por su parte, sticky empieza a "seguirte" cuando llegues a él mediante el scroll.

Display block

- **block:** Usa el 100% del ancho que tenga sin importar si el contenido tiene o no ese espacio. Ocupa siempre el 100% del width. Lo que los apila uno encima del otro.
- **inline:** Ocupa solo el espacio de su contenido. Lo que permite agruparse unos al lado del otro. No se le puede dar margen ni padding vertical pero sí horizontal. Tampoco se le puede agregar un alto o ancho.
- **inline-block:** Utiliza el espacio que ocupa el elemento. Solo el que ocupa el elemento, no el 100%.

block vs inline vs inline-block

Below are a bunch of `<div style="width: 50px"...>` with different `display` settings.



Display Flex

Flex nos permite distribuir el espacio entre los elementos/contenedores de nuestra página web, por el contrario que grid, flexbox es unidimensional, trabaja con filas o columnas, pero solo una de cada una. Para usar flex siempre debemos tener un contenedor padre el cual usará el `display flex`.

*Por defecto `flex-direction` es `row`.

***flex-wrap: wrap;** → Acomoda los elementos dependiendo del viewport width del usuario, y si no hay más espacio baja los elementos ya que no caben el ancho.

***justify-content** nos permite alinear los elementos de forma horizontal. **align-items** hace lo mismo pero verticalmente.

***box-sizing: border-box;** → Suma el padding y el borde con el width del elemento. Si aumenta el padding o el borde, el width disminuye para no generar un scroll horizontal. Lo mismo al contrario en caso de que aumente width. Esto se hace para que el elemento ocupe lo que le indicamos y no más.

Flexbox Layouts

```
.container {  
  border: 0.3rem solid black;  
  display: flex;  
  flex-wrap: wrap;  
}
```

```
.box {  
  height: 10rem;  
  flex-basis: 10rem;  
  flex-grow: 1;  
}
```

flex-grow → Toma el espacio que sobra para que no haya espacios en blanco. Al darle el valor 1 toma el espacio sobrante que deja otro elemento.

flex-basis → Define el tamaño que tendrá un elemento antes de que se distribuya al tamaño restante.

align-items: stretch; → Estira los elementos al 100% del valor de su contenedor padre. Ocupa todo el espacio vertical.

baseline → Toma solo el alto de su contenido. Lo limita a su propia altura.

order → Cambia el orden de los hijos del contenedor padre.

*Todos los contenedores que no tengan orden se pasan para la izquierda. Y los que sí lo tengan empiezan desde donde terminan los que no tienen.

Variables

Las variables nos permiten guardar un valor para evitar generar mucho código repetido.

```
:root {  
  --primary-color: #003476;  
  --secondary-color: #b4d2f7;  
  --header-size: 4rem;  
  --font: 1.8rem;
```

```
}
```

:root → Hace referencia a la etiqueta html. Aquí es donde declararemos todas nuestras variables. Para declarar una variable asignamos su nombre con dos guiones y luego su valor. Para usarla:

```
header {  
  width: 100vw;  
  height: 15vh;  
  background-color: var(--primary-color);  
}  
main {  
  width: 100vw;  
  height: 70vh;  
  font-size: var(--font);  
}  
h1 {  
  font-size: var(--header-size);  
  color: var(--primary-color);
```

Web Fonts

Las fuentes genéricas son las que vienen por defecto en nuestro sistema operativo y con las cuales se abrirán en un principio todos los sitios web. Algunas de estas son:

Generic-families

Genericas	Fuentes	
serif	Times New Roman	Georgia
sans-serif	Helvetica	Verdana
cursive	<i>Dancing Script</i>	<i>Great Vibes</i>
monospace	Courier New	Roboto Mono

Las fuentes que casi siempre se usan son:

- Light 300
- Medium 500
- Regular 400
- Bold 900

Si queremos traer fuentes externas y usarla en nuestro proyecto existen 2 formas:

1. Con **@import** desde el archivo CSS

```
@import
url('<https://fonts.googleapis.com/css2?family=Roboto+Slab:wght@100&display=swap>');
```

El cual no es tan buena práctica porque afecta el performance del sitio web. Casi siempre la mejor práctica es la segunda opción.

2. Con la etiqueta link desde HTML

```
<link
href="<https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;900&display=swap>"
rel="stylesheet"
/>
```

***display=swap** hace que el proyecto se cargue con la fuente por defecto, y una vez cargue todo el mismo, se cargará la fuente

No es bueno cargar más de una fuente externa. Debido a que cada link genera una petición y si agregamos muchos puede afectar el performance.

Para usar estas fuentes en los estilos basta con poner el nombre que nos indica [Google Fonts](#) como valor de la propiedad font-family

```
html {
  font-family: "Roboto", sans-serif;
}
```

Responsive Design

Media queries

Los media queries nos permiten hacer que nuestros proyectos sean multiplataforma mediante breakpoints. Los breakpoints son puntos donde hacemos cambios, como renderizar elementos o contenedores. Los media queries se agregan así:

```
@media (min-width: 600px) {
  .c2,
  .c3,
  .c4 {
    width: 50%;
  }

  .letra {
    font-size: 12px;
  }
}
```

Existen diferentes estrategias para trabajar el responsive. La más recomendada actualmente es la **mobile first**. Esta nos invita a desarrollar desde un dispositivo mobile y luego ir escalando a dispositivos mayores. El orden que normalmente se maneja es: mobile, tablet, desktop.

La mejor forma de trabajar con media queries es enlazando los archivos de estilos desde el head. De esta manera:

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

<link
  rel="stylesheet"
  href="/tablet.css"
  media="screen and (min-width: 600px)"
/>

<link
  rel="stylesheet"
  href="/desktop.css"
  media="screen and (min-width: 800px)"
/>
```

El primer link siempre será el mobile y no hay necesidad de especificar el atributo **media**.

Estrategias de responsive: Mostly Fluid

Este tipo de estrategia se inicia con columnas y a medida que se crecen las pantallas, las columnas van siendo acomodadas. El archivo de estilos principal (styles.css) es el archivo que usaremos para trabajar nuestra primera versión responsive, es decir, la mobile. Luego iremos desarrollando las otras como *desktop.css* o *tablet.css*

*Los media queries por buenas prácticas siempre deben ir al final de los estilos (en caso de que los tengamos en un solo archivo todos)

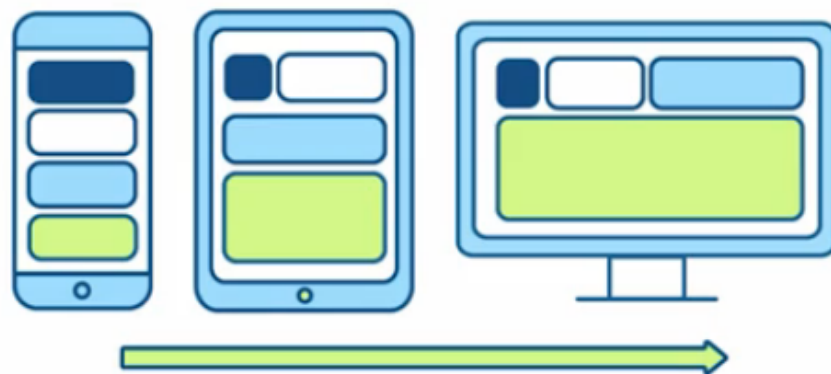
Layout Shifter CSS



Con esta estrategia o patrón de diseño responsive se notan más los cambios. Se inicia con un diseño modo vertical y a medida que se va creciendo la pantalla se empieza a acomodar los contenedores padres que se van renderizando con otros elementos dentro. La clave de este diseño es el reposicionamiento de contenedores y la colocación debajo de otras columnas.

Column Drop

Column Drop



En este patrón se tiene todo de forma vertical pero a la hora de que va creciendo el contenido empieza a arrojar los contenedores y los reposiciona a la línea principal, secundaria, etc... En este no se limita el crecimiento del container porque queremos que se estire por completo, que ocupe todo el ancho de la pantalla.

Enlace para profundizar en patrones de diseño web adaptable

Buenas prácticas y ejemplos de responsive

Existen muchos patrones de diseño responsive, algunos hasta incluyen JS pero los 3 vistos anteriormente son los más usados y reconocidos.

Algunas de las buenas prácticas que debemos implementar a la hora de desarrollar en responsive son:

- Siempre preparar los archivos CSS por breakpoint y no todos en una sola hoja de estilos.
 - mobile.css/style.css
 - tablet.css
 - desktop.css
- Trabajar con máximo 6 breakpoints. Debido a que se generará más código lo cual impactará en performance y no es buena práctica.

Imágenes responsive

`<picture>` y `<source>` nos ayudan para trabajar imágenes responsive.

```
<picture>
  <source media="(min-width:1300px)" srcset="/imgs/large.jpg" />
  <source media="(min-width:1000px)" srcset="/imgs/medium.jpg" />
  
</picture>
```

la etiqueta `<source>` recibe como parámetros *media* y el *srcset*. *media* recibe los valores en los cuales se van a mostrar diferentes imágenes dependiendo del tamaño de la pantalla. Y *srcset* recibe la ruta de la imagen y el navegador decide cuál cargar dependiendo del peso. *srcset* se comunica con el navegador y le da opciones para escoger la mejor imagen para ese dispositivo. Recibe una imagen como parámetro.

Herramientas para responsive

mydevice nos muestra los viewport de todos los dispositivos para saber con cuáles podemos trabajar. Pero por lo general con trabajar en: 300px, 600px y 1040px tenemos varios dispositivos optimizados.

mydevice.io : [web devices capabilities](#)

[A Web developer's browser - Responsively App](#)

Accesibilidad

Garantizar la accesibilidad de nuestro sitio es hacerlo inclusivo. Permitir que personas con discapacidades puedan acceder y disfrutar el contenido de una forma fácil para ellos.

Semántica

Debemos usar HTML semántico y no solo etiquetas div. Esto le permite a los software de lectura de pantalla saber donde están ubicados para poder guiar a su usuario. Las etiquetas semánticas son las mostradas en la clase **index y su estructura básica: body**

Textos

Se recomienda siempre usar medidas relativas, como rem, para los textos. Esto con el fin de que personas con discapacidades visuales puedan aumentar el tamaño de la fuente desde su navegador. Si el texto estuviese en pixeles no se podría cambiar.

Labels, alt y title

Los **labels** funcionan junto a los inputs. Es importante porque al envolver el input en un label, este le da un foco especial que permite hacer click sobre el label y nos lleva a rellenar la información del input. Le ayuda mucho a algunos software de lectura a enfocarse en el input directamente.

El atributo **alt** nos ayuda a leer la descripción de las imagenes para personas con discapacidades visuales. También sirve para agregar un texto alternativo en caso que la imagen no se cargue.

El atributo **title** puede ser usado en varias etiquetas HTML como **img** o **a** y su función es agregar una descripción para cuando el usuario haga hover sobre algún elemento.