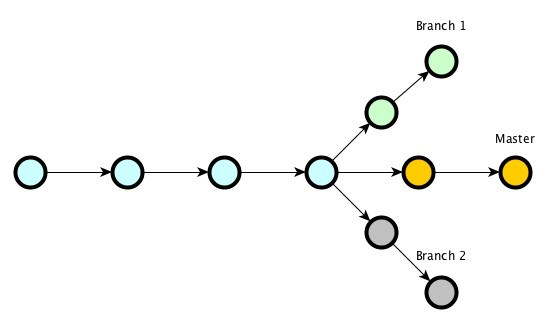
Git, editores de código, HTML y CSS

# Recapitulando

Si nos fue bien con el taller 2 entonces ya estamos entendiendo cómo funciona Git. El proceso para crear un repositorio parte en nuestro computador donde creamos una carpeta que contiene nuestro proyecto (también lo conoceremos como **repositorio local**). Dentro de la carpeta tendremos nuestro código fuente (en este caso un archivo HTML) y dentro de esa carpeta iniciamos un repositorio git con el comando git init. Vale, ahora vamos a la web. Creamos un repositorio (también lo conoceremos como **repositorio remoto**). Y ejecutamos los comandos que nos entrega Github luego de crear el repositorio remoto. Estos comandos son siempre los mismos en la primera subida a nuestro repositorio.

Importante saber que Git nos permite crear distintas versiones de nuestros proyectos. Esto lo hacemos creando **ramas o branchs,** cuando creamos una rama hacemos una copia exacta de nuestro proyecto en un determinado momento y trabajaremos sobre la rama sin alterar la versión de la cual obtuvimos la rama. En dibujo es algo así:



Donde cada nodo o círculo es un commit (ya veremos qué es un commit) y la flecha es una rama o branch que se extrajo de la rama principal o **master.**

# ¿Por qué saber Git?

Una pregunta importante que no he resuelto es para qué aprender esto. Si es que vale realmente la pena aprenderlo. La respuesta es… sí. Git hoy en día es la mejor forma para gestionar versiones de código y lo mejor de todo es que usando la plataforma Github podemos acceder a otros proyectos de programadores de todo el mundo, podemos revisar cómo están construidas algunas aplicaciones y tomar códigos para aplicar en nuestros propios proyectos, aprender, subir proyectos que puedan servir a otros programadores. Si revisan el video que les envié sobre Git verán que es una red social de programadores.

# Los comandos de Git

Como habrán notado, Git tiene un varios comandos que nos permiten realizar diferentes acciones de acuerdo al nombre del comando. Si recuerdan al instalar git escogimos la opción que nos permite ejecutar comandos de git en nuestra línea de comandos. Todo comando git comienza con la palabra *git* luego, el nombre del comando y luego los parámetros que recibe el comando. Esta sintaxis básica siempre la encontraremos en comandos que ejecutamos en la consola.

Recuerden siempre estar familiarizados con la consola, ya que como computines que somos, siempre nos enfrentaremos a esta y sobre todos si nos dedicamos al desarrollo de apps.

Comando **status**, para ver los cambios que ha sufrido nuestro repositorio.

Debemos conocer el comando **add**. Este comando nos permite agregar archivos al commit que haremos. Esto es útil cuando queremos excluir algunos archivos del commit que haremos (luego veremos qué es un commit). Podemos indicar al comando que agregue todos los archivos que sufrieron cambios.

Asi también debemos conocer el comando **commit**, commit significa “cometer” y en este contexto es “confirmar los cambios que vamos a subir”. Con esto creamos una “bandera” (o *flag* para los pro) que tendrá un identificador único para identificar un conjunto de cambios en ciertos archivos. Este comando nos permite agregar un mensaje con tal que sea descriptiva la identificación de la subida.

Debemos conocer los comandos **push** y **pull**. Cuando hablamos de git estos términos salen muy a menudo. **Push** significa empujar, hace referencia a “empujar” o subir nuestros cambios. **Pull** significa jalar o tirar, hace referencia a “jalar” cambios desde un repositorio remoto.

Otro comando importante es **clone**, nos permite clonar un repositorio remoto hacia nuestro computador.

En Github podemos ver el historial de las subidas identificadas por commit junto a su descripción.

Más adelante verán que se pueden hacer varios commits en una subida.

### 

## Entonces, veamos algunos comandos básicos de git.

#### Para iniciar un repo local

> git init

Este comando no recibe ningún parámetro.

#### Para agregar archivos al commit:

> git commit -a

o

> git commit -m “mensaje descriptivo del commit”

Este comando puede recibir los parámetros -a o -m. El -a nos sirve para crear un commit sin descripción y el -m nos sirve para indicar que es un commit que tiene mensaje y se indica inmediatamente después del parámetro -m (entre comillas).

#### Para subir nuestros cambios:

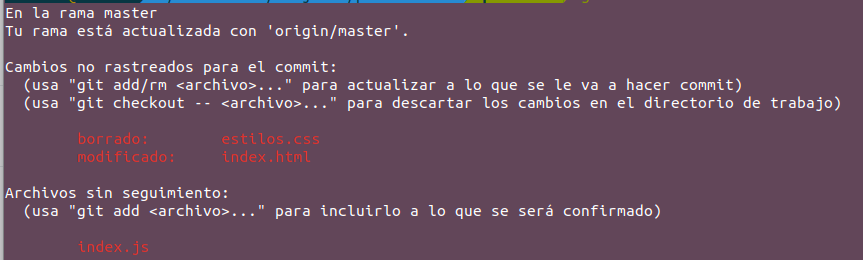
> git push

Cuando hacemos un push por primera vez debemos ejecutar este comando con el parámetro -u origin master. Esto indica que vamos a subir a la rama master.

#### Para ver el estado de nuestros archivos en nuestro repositorio local

> git status

Aquí un ejemplo de lo que nos retorna el git status, donde borré un archivo (estilos.css), modifiqué el index.html y agregué un archivo llamado index.js



##### Para clonar un repositorio

> git clone [nombre\_del\_repo]

Este comando recibe como parámetro el nombre del repositorio. Es importante que desde nuestro terminal estemos ubicados en la carpeta que contiene nuestro repositorios. Cuando hacemos un clone, se descarga una carpeta con el contenido del repo descargado (o clonado)

## Más comandos y descripciones

Aquí podrán ver una tabla con los comandos básicos para empezar a usar Git:

<https://confluence.atlassian.com/bitbucketserver/basic-git-commands-776639767.html>

## Info oficial

Y aquí una lista con todos los comandos

<https://git-scm.com/doc>

# IDE’s y Editores de código

Como sabrán existen muchas plataformas para codificar.

Por un lado tenemos los famosos IDE (entorno de desarrollo integrados) que como su nombre indica integran un editor de código, depurador, compiladores, intérpretes, herramientas para conectar bases de datos, configurar conexiones a servidores, ver métricas de la ejecución de la aplicación que estamos desarrollando, etc.

Por otro lado, tenemos los editores de texto, que son sistemas que, inicialmente, sólo eran interfaces que nos presentaban el código fuente de una forma legible y bonita (colores, formato, indentaciones) y quizás algún terminal para ejecutar nuestras aplicaciones.

Hoy en día los editores de código han potenciado sus funcionalidades, principalmente, gracias a los plugins o complementos que podemos añadir a estos. Estos nos ayudan a escribir código por medio de sugerencias, identificar el lenguaje en que estamos escribiendo, analizar la calidad de nuestro código, integrar utilidades de versionado de código con Git, ordenar nuestro código para mejor lectura, etc.

Entonces, ¿cuándo usar uno u otro?. Como todo en la vida depende. Depende de lo que vamos a desarrollar o codificar. Si nuestro sistema es un tanto complejo, si es para una plataforma en particular y por temas de licencia sólo podemos usar un IDE específico, si requerimos conectarnos a una base de dato, o si requerimos “levantar” toda una infraestructura para ejecutar nuestra aplicación, etc. Pero si nuestra aplicación no es tan compleja y sólo requerimos “levantar” una parte de la aplicación, seguramente nos convenga más trabajar con un editor de código ya que su ejecución es mucho más rápida, su uso y configuración son mucho más sencillos.

Algunos ejemplos de IDE’s:

* Netbeans
* Eclipse
* Android Studio
* Visual Studio
* IntelliJ

Algunos ejemplos de editores de código:

* Visual Studio Code (recomendado)
* Sublime Text
* Atom
* Notepad++

Para nuestros propósitos usaremos un editor de código, les recomiendo Visual Studio Code, Sublime Text o Atom. Y ¿por qué? Bueno, por las mismas razones que les comentaba anteriormente. Para nuestros propósitos; construir html’s, hojas de estilos, códigos Javascript y levantar servidores web, es más que suficiente.

## Extensiones

Con estos 3 editores de código podemos instalar plugins que son útiles al facilitarnos la vida cuando escribimos código. Para este módulo usaremos algunos que nos ayude a escribir HTML, CSS Y javascript. Los más importantes dentro de estos lenguajes están:

* Emmet (<https://emmet.io/>)
* Live server (en VS Code) o LiveReload (en Sublime Text)
* Beautify o Prettify (<https://marketplace.visualstudio.com/items?itemName=HookyQR.beautify>)
* Lorem Ipsum
* GitLens (en VS Code) o GitSavvy (en Sublime Text)

**Emmet**, como su sitio dice, es un complemento para muchos editores de texto populares que mejora enormemente el flujo de trabajo HTML y CSS. O sea, nos permite escribir HTML y CSS de forma más rápida por medio de atajos y autocompletado.

**Live Server**, como su sitio dice, en Visual Studio Code levanta servidor de desarrollo local con función de recarga en vivo para páginas estáticas y dinámicas. O sea nos sirve para visualizar páginas y cada vez que hacemos un cambio en nuestro código la página se recarga mostrando los cambios realizados.

**LiveReload**, es un complemento para hacer recargas de página del navegador web para el editor Sublime Text 3.

**Beautify** y **Prettify**, nos permite formatear y ordenar código Javascript, JSON, CSS y HTML en Visual Studio Code y Sublime, respectivamente.

**Lorem Ipsum**, es un complemento para VS Code y Sublime Text, nos permite generar un texto genérico de forma rápida.

**GitLens** y **Gitsavvy**, contiene utilidades para trabajar fácilmente con Git.

# 

# HTML, más al detalle

## Estructura básica de un html

Tal como vimos en el taller 2, un archivo html debe contener una estructura básica predefinida; todo archivo html debe empezar con la etiqueta *DOCTYPE* que nos indica que el presente archivo será un archivo de tipo html y luego indicar la etiqueta html.

HTML es un lenguaje que trabaja con etiquetas (también conocidas como tags), toda etiqueta empieza con el símbolo < acompañado del nombre de la etiqueta y luego, si la etiqueta permite contener otras etiquetas acompaña el símbolo >, de lo contrario si la etiqueta no permite contener otras etiquetas debe acompañar el símbolo / luego el >

## Apertura y cierre de etiquetas

Hay una etiqueta llamada *div* que **sí permite** (por reglas de de sintaxis de HTML) contener otras etiquetas, por ende, para abrir una etiqueta escribimos: *<div>* dentro podemos escribir otras etiquetas, y luego cerramos la etiqueta de la siguiente manera: *</div>.*

Por otro lado tenemos la etiqueta *input* que **no permite** (por reglas de HTML) contener otras etiquetas, por ende, cuando escribimos una etiqueta input la debemos cerrar en la misma etiqueta de apertura, de la siguiente manera: *<input />*

## 

## ¿Cómo es una estructura básica de un archivo html?

Cómo vimos en el anterior material, y tomando en cuenta la apertura y cierre de etiquetas, **todos** nuestros html tendrán una estructura básica:

|  |
| --- |
| <!DOCTYPE html>  <html>  <head>  <title>Document</title>  </head>  <body>    </body>  </html> |

*index.html*

Esto nos indica que nuestro archivo html tiene dentro una sección head y una sección body. Dentro de la sección head tenemos una etiqueta title.

Dentro de la etiqueta *head* indicamos toda la información que corresponde a contenido **no visible** para el usuario. Por ejemplo aquí indicamos la codificación de caracteres (para poder usar tildes y caracteres extraños), además podemos indicar si nuestra página permite que se le pueda hacer zoom (en dispositivos móviles), podemos definir el título que se muestra en la pestaña del navegador cuando se cargue nuestro html, podemos definir estilos visuales, podemos definir algunos scripts escritos en Javascript, etc.

Dentro de la etiqueta body incluimos todo el **contenido visible** de nuestra página. Aquí incluimos los textos, menúes, cabeceras, pies de página, formularios, imágenes, animaciones, etc.

## Atributos en html

El lenguaje HTML define algunos **atributos** a ciertas etiquetas, por ejemplo, la etiqueta <html> tiene el atributo llamado *lang* que nos permite indicar el lenguaje de nuestro html. Esto es útil para que nuestro navegador lo detecte y los buscadores de internet puedan organizar los sitios de acuerdo al idioma que corresponde.

Todos los nombres de atributos deben ir después del nombre de la etiqueta y su valor debe ir entre comillas. Siguiendo el ejemplo del tag html y su atributo lang establecido con valor español, debe escribirse así:

|  |
| --- |
| <html lang=”es”> |

Además, podemos agregar más de un atributo para una etiqueta.

## Etiquetas en HTML

Como vimos en el anterior material, las etiquetas básicas de html son:

html

head

body

La etiqueta *head* permite contener otras etiquetas, las más comunes son las siguientes:

* meta: Para indicar metadatos de nuestra página (datos sobre nuestra página)
* title: Título de nuestra página (aparecerá en el nombre de la pestaña)
* style: Para escribir estilos visuales usando lenguaje CSS
* script: Para escribir scripts en lenguaje Javascript

Y dentro de la etiqueta body tenemos un mundo amplio de etiquetas:

main, section, aside, header, footer, article, div, input, button, time, a, p, h1, h2, h3, h4, h5, h6, hr, ul, ol, li, i, audio, video, etc.

# Un paso más allá

Ahora veamos un ejemplo en html para entender la estructura que va dentro del body de un sitio:

|  |
| --- |
| <!DOCTYPE HTML>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />  <title>Your Website</title>  </head>  <body>    <header>  <nav>  <ul>  <li>Your menu</li>  </ul>  </nav>  </header>    <main>  <section>    <article>  <header>  <h2>Article title</h2>  <p>Posted on <time datetime="2009-09-04T16:31:24+02:00">September 4th 2009</time> by <a href="#">Writer</a> - <a href="#comments">6 comments</a></p>  </header>  <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.</p>  </article>    <article>  <header>  <h2>Article title</h2>  <p>Posted on <time datetime="2009-09-04T16:31:24+02:00">September 4th 2009</time> by <a href="#">Writer</a> - <a href="#comments">6 comments</a></p>  </header>  <p>Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.</p>  </article>    </section>  </main>  <aside>  <h2>About section</h2>  <p>Donec eu libero sit amet quam egestas semper. Aenean ultricies mi vitae est. Mauris placerat eleifend leo.</p>  </aside>  <footer>  <p>Copyright 2009 Your name</p>  </footer>  </body>  </html> |

Conen este repo <https://github.com/a-troncoso/taller-apps-intenet>, y en el archivo *2-html-etiquetas-body.html* verán el archivo con comentarios.

Luego de descargado abran el archivo html con su navegador y verán que nos muestra texto plano con un ningún formato

## Definiendo las secciones básicas de la página

Antes de leer esto revisen bien (desde el editor de código y el navegador) el HTML anteriormente citado. Vean la estructura dentro del body y cómo ésta se visualiza en el navegador (usando live server).

Si se fijan aparecieron varias etiquetas que cumplen distintas funciones. Nombremos algunas:

* **<header>:** Define una parte que se usará como cabecera de una sección.
* **<main>:** Define la sección principal de nuestra página.
* **<section>:** Como su nombre lo indica, define una sección dentro de nuestra página.
* **<aside>:** Define una sección que contiene información secundaria.
* **<footer>:** Define el pie de la página o sección.

En la siguiente tabla (obtenida de MDN) podemos ver los tags (o etiquetas) que definen secciones en una página:

|  |  |
| --- | --- |
| **Elemento** | **Descripción** |
| [**<body>**](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/body) | Representa el contenido principal de un documento HTML. Solo hay un elemento <body> en un documento. |
| **<section>** | Define una sección en un documento. |
| **<nav>** | Define una sección que solamente contiene enlaces de navegación |
| **<article>** | Define contenido autónomo que podría existir independientemente del resto del contenido. |
| **<aside>** | Define algunos contenidos vagamente relacionados con el resto del contenido de la página. Si es removido, el contenido restante seguirá teniendo sentido. |
| [**<h1>,<h2>,<h3>,<h4>,<h5>,<h6>**](https://developer.mozilla.org/en-US/docs/HTML/Element/Heading_Elements) | Los elemento de cabecera implementan seis niveles de cabeceras de documentos; <h1> es la de mayor y <h6> es la de menor importancia. Un elemento de cabecera describe brevemente el tema de la sección que introduce. |
| **<header>** | Define la cabecera de una página o sección. Usualmente contiene un logotipo, el título del sitio Web y una tabla de navegación de contenidos. |
| **<footer>** | Define el pie de una página o sección. Usualmente contiene un mensaje de derechos de autoría, algunos enlaces a información legal o direcciones para dar información de retroalimentación. |
| [**<address>**](https://developer.mozilla.org/es/docs/Web/HTML/Elemento/address) | Define una sección que contiene información de contacto. |
| **<main>** | Define el contenido principal o importante en el documento. Solamente existe un elemento <main> en el documento. |

Hasta aquí hemos visto elementos que definen ***secciones*** dentro de una página, pero no todo en HTML son secciones 😯. Hay etiquetas para otros propósitos también:

* Definir grupos de contenido
* Definir semántica de textos
* Incrustar contenido
* Tabular contenido

Por favor revisen las tablas que se muestran en el sitio de MDN donde se muestran las etiquetas disponibles para los distintos propósitos.

## Documentación ~~casi~~ completa:

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

# ¡Y por último! Conozcamos a don **CSS**

Hola, soy CSS:

.App {

display: grid;

justify-content: center;

margin: 100px;

background-color: bisque;

}

.App p {

background: lightgray;

box-sizing: border-box;

color: darkgray;

font-family: Arial, Helvetica, sans-serif;

padding: 16px 8px;

}

CSS es una sigla de Cascading Style Sheets. Nos sirve para darle estilo a nuestra paginas 😎. O sea, nos sirve para indicar cómo los elementos HTML de la página son desplegados en ella y qué apariencia tienen.

## ¿Y cómo lo usamos?

CSS define una serie de reglas para aplicar estilo a nuestros HTML. Estas reglas son un conjunto de propiedad/valor CSS. Cada regla indica un estilo visual que contiene nuestro elemento HTML.

Cabe señalar que a **todos** los tags que van dentro del <body> de nuestro HTML les podemos aplicar reglas CSS.

## ¿Cómo aplicamos CSS?

Tenemos 3 formas de aplicar CSS en nuestros archivos HTML:

1. En una etiqueta HTML (estilo en línea)
2. Dentro de la etiqueta <style> (hoja de estilo interna)
3. En un archivo con extensión CSS (hoja de estilo externa)

### Estilo en línea

Podemos indicar reglas CSS en una etiqueta CSS, de la siguiente manera:

<body style="background-color: blue">

Podemos indicar más de un estilo a una etiqueta, las separamos con punto y coma

Por ejemplo queremos que el body de nuestro HTML tenga un color de fondo azul y que todos los textos dentro del body sean de color celeste:

<body style="background-color: blue; color: cyan">

### 

### Estilo dentro de la etiqueta <style>

Podemos indicar propiedades CSS dentro de la etiqueta style, que a su vez **va dentro del <head> de nuestro HTML**. Veamos lo mismo que hicimos anteriormente (body con color de fondo azul y texto celeste) aplicando dentro de la etiqueta style:

<head>

<!-- etiquetas meta, link, script, etc… -->

<style>

body {

background-color: blue;

color: cyan;

}

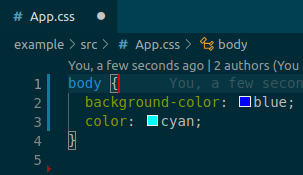
</style>

</head>

### 

### Estilos dentro de un archivo con extensión CSS

También podemos incluir CSS dentro de un archivo con extensión CSS:



Cabe mencionar que en nuestro CSS no definimos ninguna cabecera o algo así como un DOCTYPE ni nada extraño, es sólo empezar a escribir el selector CSS y ya empezamos a poner las propiedades (o atributos) junto a sus valores separados con punto y coma.

Ahora que el CSS está fuera de nuestro archivo HTML, ¿cómo los conectamos?.... Pues, debemos llamar al archivo CSS desde el HTML usando una etiqueta especial que **va dentro del <head> de nuestro HTML:**

<head>

<!-- aquí van etiquetas <meta>, <script>, etc… -->

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

</head>

En este caso tenemos un archivo llamado App.css en la misma carpeta que nuestro archivo HTML.

#### 

#### Entonces, ¿Cuál de estos 3 métodos usaremos?

**El tercer método. Fin.**

El método CSS inline y CSS dentro de etiqueta <style> se considera que “ensucian” el código HTML al incrustar código que no corresponde al HTML. Recordemos que siempre tratemos de que los archivos contengan un sólo lenguaje, sin mezclar unos con otros. Dicho de otro modo, evitemos incrustar lenguaje CSS dentro del lenguaje HTML (así también ocurrirá con Javascript que veremos en el futuro).

## 

## Más cosas a saber

Si se fijan nosotros definimos nuestros estilos para el body, pero, ¿Y si queremos definir estilos para otras etiquetas?

CSS trabaja con una cosa llamada **selectores,** los selectores nos sirven para indicar a quién le estamos aplicando estilos.

En nuestro ejemplo le estamos aplicando los estilos al <body>, claramente no sólo podemos aplicar al body, si no también a cualquier etiqueta hija de body (header, nav, main, section, article, h1, h2, p, etc).

Entonces, hasta el momento sabemos que desde el archivo CSS accedemos a las etiquetas HTML por su nombre de etiqueta. Ejemplos:

#### Aplicando estilos a <header>:

header {

display: block;

background-color: #c3c3c3;

border-style: solid;

border-width: 1px;

border-color: #7a7a7a;

}

#### Aplicando estilos a <main>:

main {

display: grid;

background-color: #fff;

}

## Más selectores

La verdad es que hay muchos selectores, veremos los más importantes.

* Selector por clase
* Selector por elemento
* Selector por nombre de clase y elemento
* Y otras más

### Selector por clase

Las forma más comunes para aplicar estilos a una etiqueta HTML es usando clases. Veamos cómo definimos una clase en CSS:

.una-clase {

display: inline-block;

color: red;

}

*estilos.css*

¿Cómo lo aplicamos en HTML?

Así:

<div class="una-clase">

*<!-- contenido dentro del div... -->*

</div>

*index.html*

Con esto definimos una clase en nuestro archivo llamado estilos.css. La definimos usando **un punto antes del nombre de la clase**. Algunas cosas importantes al definir una clase CSS:

1. El nombre de la clase empieza con un punto
2. El nombre de la clase debe ser descriptivo, de acuerdo a dónde se va aplicar (por ejemplo el nombre de clase para el <main> podría llamarse .*contenedor-principal)*
3. El conjunto de reglas CSS debe estar dentro de llaves ({ propiedad: valor; ...})
4. Las propiedades y valores deben estar separadas entre sí por punto y coma
5. Por lo general, se recomienda que el nombre de las clases que tienen más de una palabra, se separen con guión medio.

Luego, en el HTML debemos agregar el ***atributo*** llamado **class** que tendrá como valor el nombre de la clase creada en HTML.

Algunos puntos importantes respecto a clases en HTML:

1. Todas las etiquetas que van dentro del <body> soportan el atributo **class**
2. Podemos aplicar más de una clase a una etiqueta, simplemente separando el nombre de las clases con un espacio (asumiendo que en nuestro CSS tenemos una clase llamada contenedor-principal y otra clase llamada mobile):

<main class=”contenedor-principal mobile>...</main>

### Selector por elemento

Esta es la que vimos en primer lugar. Es cuando por medio de CSS seleccionamos los tags HTML por su nombre.

body { … }

header { … }

main { … }

### Selector de todos los elementos HTML

Este es un selector especial, no sirve para seleccionar **todos** los elementos del html. Se seleccionan por medio del carácter asterisco. Ejemplo:

\* {

margin: 0;

padding: 0;

}

Con esto estamos diciendo que todos los tags dentro del HTML tienen margen y padding 0 (luego veremos las propiedades CSS para entender qué son los margenes y paddings).

### Selector por nombre de clase y elemento

Podemos seleccionar elementos HTML en CSS combinando nombres de clases y nombres de tags. Ejemplo:

p.resaltado {

font-weight: bold;

}

Con esto estamos seleccionando todas las etiquetas <p> y que a su vez tengan la clase llamada resaltado. Por ejemplo, la siguiente clase estaría seleccionada (y con font-weight: bold le estamos diciendo que sea un texto en negrita)

<p class="resaltado">Lorem ipsum dolor sit amet, consectetur</p>

# Más selectores

Visiten el siguiente sitio para ver todos los tipos de selectores

<https://www.w3schools.com/cssref/css_selectors.asp>