**HTML CSS by freeCodeCamp**

**HTML 🡪** It is a markup language, not programming language. It is responsible for webpage structure:

Syntax: **<element>** content **</element>**

* Install Chrome, Install VS Code, some extensions too
* Create a folder, open it in VS Code and add a HTML file
* Una vez en VS Code asociamos el Repository Git: https://github.com/PadillaTom/FrontEndJourney.git

(Log in user.name // user.email // git init // git remote origin LINK //git pull origin master // git push –u origin master)

* Una vez Asociado Git con Remote y Folder comenzamos los tutoriales

**Structure and Elements:**

<!DOCTYPE html> 🡪 HTML version.

<html> 🡪Root Element.

<head> 🡪 Information about the page (Meta, Links, Title, etc) .

Content won’t be visible.

<title> xxx 🡪 Page title (shows in tab).

<body> 🡪 What will be displayed on the page.

* There are Parent Tags and Children Tags:  
  <head>

<title> **Titulo 1**

* If something is too long we can use **ALT + Z** (word wrap)
* What is Emmet? Something that speeds up my work. In VS Code it’s already built in. EJ: Every time I want to create a HTML element I don’t need to type the < >, Emmet will fill with the suggested abbreviation. <html></html>
* Vemos la estructura general y una vez armada podemos comenzar: Se recomienda utilizar Live Server (right click on VS Code + open with Live Server)

**Headings <h>:**

They are a kind of title. There are six types, being h1 the bigger size:

<h1> <h2> <h3> <h4> <h5> <h6>

**Paragraph <p>:**

Where we are going to put some kind of Text component. HTML it is “white space collapsing” (HTML is going to ignore extra spacing) : <p> **Hello**

**this is a paragraph**</p> 🡪 HTML will ignore all the spaces and print it all together.

* Si queremos crear SPACES tenemos que usar un tag especial
* Por momentos sabemos que tenemos content, pero no lo tenemos, podemos usar **LOREM X** (x = number of words):

<p> **Lorem 20** </p>

**Images <img src = “ “ alt=” “ >:**

We can upload images for every kind of content. They must be in the same folder as the website or can be uploaded by URL().

SRC = ” “ 🡪 Path (Folder **./** or URL() )

ALT = “ “ 🡪 What we can see when the image is broken, like a short description

./ 🡪 “in the same directory”

../ 🡪 “outside the directory”

URL() 🡪 Get to the image file (Jpg, Png, etc) and copy link.

Debemos usar website de busqueda para copyright free images (Pixabay, Gratisography, etc).

* Render size 🡪 via HTML (we will use CSS or even cropping

them before)

<img src=”” width=”260 alt=”””> 🡪 Width =” “ // Heigth will be

automatically adjusted.

* Ahora … si tenemos muchas imagines el usuario va a experimentar un website muy lento…. Podríamos ajustar las imágenes previamente Photoshop, Mac, Etc. o **utilizar CSS.**
* Creamos una IMG folder y dentro sub folders según utilidad.

**Comments <!—xxx -->:**

We should comment at the Start or End of every section.

**CTRL + }** 🡪 To add or convert into a comment.

**Line Breaks <br>:**

This will break the white space collapsing. Just like print(“ “) in python. It adds lines.

**BR\*6** 🡪 Shortcut that will add 6 times the <br>

**Links <a href=” “>:**

It is called an Anchor Tag:

<a href= ” # ” > **Some clickable Text** </a> 🡪 **#** it is a placeholder that will be replaced later in time for a real link

* Target =” **\_blank** “ 🡪 It will open link on a new Tab
* What about internal surfing? Creamos un **About.html** y agregamos un **<a>** que nos lleve a **Index.html**.
* “Back to the top” 🡪 Usamos **src =” # “. #** Como en CSS # representa un ID, podríamos poner cualquier **#ID** y nos llevaría hasta él.
* Podemos usar distintos elementos para clickear un Link:

<a>

<img>

**<Sup> and <Sub> Elements:**

These elements will be places in top or bottom of the RENGLON.

<h1> **Hello I am** <sup>**1st**</sup> **John to arrive in** <sub>**town**</sub></h1>

* 1st and Town will be shown NOT IN LINE.

**<Strong> and <Em>phasis Elements:**

Within some text we can have STRONG and EM.

<p>**lorem20**<strong>**Bold Text**</strong><em>**Italic Text**</em></p>

Strong 🡪 Bold text

Em 🡪 Italic Text

* The ideal is to have a CSS for style and HTML for structure. Not to mix them both.

**Special Characters” &char; “ :**

Por ejemplo el logo de Copyright:

<h1> Copyright **&copy;** </h1> 🡪 Replace copy for another character, Emmet will suggest the names.

**List Structures <ul> <li>:**

<ul> 🡪 Unordered List: Uses points, can be removed by CSS

<ol> 🡪 Ordered List: Uses numbers, CSS can remove

<li> 🡪 List Item

* Usually we have List items inside the Lists, we can have as many as we need
* We can have Elements inside List Items

<li> <a href =” “ > **About** </a> </li>

<li> <img src=” “> </li>

* We can also have “Nested Lists”

<ul>

<li> **John** </li>

<li> **Peter**

<ul>

<li> **Html 🡪** Appears like a sub list of Peter.

<li> **Css**

</ul>

</li>

</ul>

**Tables <table> <tr> <th> <td>:**

Like any other table we need:

<tr> 🡪 Table row

<th> **Name**  🡪 Table Headings

<th> **Age**

<tr>

<td> **John** 🡪 Table Data

<td> **Peter**

<tr>

<td> **20**

<td> **25**

**Forms:**

**Input and Submit <form> <input> <button>:**

Generalmente las usamos para colectar data. HTML trabaja únicamente con el Front End, no podría coleccionar data, simplemente armar la estructura **(FormSpree website will créate the form and send info to our email)**

<form action =”” method=””>

<input type=”text” name=”” id=” ID ”>

🡪 Type: tipo de input // name: lo que se pide

<button type=”submit”> **Submit**

<input type=”submit” value=””> 🡪 Funciona como botón de submit // Value: Ingresamos texto del boton

* A su vez podemos especificar la info solicitada antes de la casilla del input

<label for=” ID “> **First Name** </label>

<input type=”text” id=” ID “>

🡪 La ID debe coincidir con la ID del input. Ejemplo: NAME. Si se hace click en la Label se resalta el casillero de Input

* Solicitar una password sin usar labels:

<p> Password

<input type=”password” placeholder=”Type in your password”>

🡪 When we type password it is NOT Visible

**Textarea Radio Checkbox <textarea> <input> <select>:**

<**textarea** name =”” id=”” cols=”10” rows=”10”>

🡪 Es simplemente un área para escribir, podemos especificar cantidad de renglones. Cols y Rows : Cantidad de renglones.

<input type=”**radio**” value=”Javascript“> **Javascript**

<input type=”**radio**” value =”Python”> **Python**

**🡪** You can ONLY select ONE of the items. **Si bien uno selecciona lo que diga, el valor real que se envia es el VALUE.** Name debe concidir entre las 3, sino podrá elegir el usuario entre varias opciones.

<input type=”**checkbox**” **checked** value=” ”>

🡪 You can select MULTIPLE values within the options. **Si bien uno selecciona lo que diga, el valor real que se envia es el VALUE.** Checked = Have the box already checked by default

<**select** name=”” >

<**option** value=” ”> Javascript </option>

<**option** value=” ”> Python </option>

</select>

🡪 You can select ONE of the list. **Si bien uno selecciona lo que diga, el valor real que se envia es el VALUE.**

**Prettier and format:**

We can install an extensión to use the formatting function: Prettier Code 🡪 Whenever I Save it will automatically format it to make it prettier and understandable.

File -> Preferences -> search for : Format On (type/save)

**Keyboard Shortcuts:**

Look the JPG for windows shortcuts.

Most important:

* Ctrl + Z : Undo
* Alt + Click : Multiple Cursors
* Ctrl + I : Select whole Line
* Alt + <- or -> arrows : Navigate Tabs
* Ctrl +up down arrows : Start or End document
* Ctrl + } : Comment Line or Start a comment
* Shift + Alt + up down : Copy line
* Alt + up down: Move the line up or down
* Ctrl + Enter : Creates a new line
* Para H Refs = “ ../ “ : Buscar en directorios

**First HTML Project DONE!!!**

**CSS 🡪**  Cascade Style Sheet. We need to create a new file with the extension **.css.** It is responsible for styling the web, the looks.

Selector **{**property: value; property: value, etc. **}**

* There are different ways of using CSS:
  + Inline CSS 🡪Inside our HTML <h1 style=” “>
  + Internal CSS 🡪 Inside HEAD: <style> h1{ ; }
  + External CSS 🡪 Inside HEAD: <link rel =” “

href= “Style.css”>

* Lo major esc rear un Style.css y asociarlo con LINK al head element de las pages.
* **Borramos todo lo anterior!!!! (Tests de los diferentes CSS)**

**Syntax:**

H1 {

Color: red;

}

**H1 🡪** Selector

**{ } 🡪** Declaration block

**Color:** 🡪 Property

**Red** 🡪 Value

**All together 🡪** Css Rule

**Selectors:**

* Element Selectors:

We should use the name of the element as the selector

h1 {Color;}

h2 {Color;}

p {Color;}

Selector Grouping:

Podriamos simplemente usar el **<body>** siendo el elemento que contiene todo. 🡪 Body {color}

Tambien podemos usar “ **,** “ 🡪 h1, h2 {color}

* ID Selectors:

Damos un ID al elemento en el html y luego lo usamos en el css. IDs should be unique, one per element.

<h1 id = “heading”> 🡪 #heading {}

* CLASS Selectors:

Las clases pueden ser compartidas de manera de hacer un estile a la class y que se aplique a todas a la vez.

<h3 class = “Green”> 🡪 .Green {}

* Podemos combinar ID junto con CLASS y una segunda CLASS:

Todos los TITLE tendrán cierta fontsize, color verde y además podemos hacer lowercase al text.

<h3 id=”title” class=”Green lowercase”>

**#title** {font-size} **.green** {color} **.lowercase** {text-transform}

**Div and Span elements <div> <span>:**

Cuando trabajamos con mucho contenido nos conviene usar Div y Span. Se usarán para agrupar a la hora de dar style.

<div> 🡪 Tag de DIV, agrupamos h3 y p

<h3> dentro.

<p>

</div>

<h3>

<p> **blablablá** <span>**blablablá**</span>

Style:

div {color; bgcolor} 🡪 Everything inside the div will be changed.

Span {text-transform} 🡪 Everything within span will change.

Podemos mezclar DIV o SPAN con CLASS tambien:

<div class=”red”> 🡪 Entonces se estiliza como **.red**

<span class=”red”> 🡪 Se le aplica la **.red**

* DIV is for starting a new line. “Block level element”
* SPAN inline style. “Inline element” 🡪 To style something within a text already, so we don’t want to create a new line.

**Inheritance:**

Whatever style we apply to a Parent element in HTML, children element are going to inherit. Unless we specifically style the children element.

* <body><p> **blablá** </p><p> **blabla** </p></body>

\*\* body {color: red} 🡪Color will be applied to both P

* <body><div><p> **blablá** </p><p> **blabla** </p></div></body>

\*\* Body {color: red} div {color: blue} 🡪They will turn blue, since DIV is a child element and it was specifically styled to blue.

* <body><div><h2>**Hello**</h2><p> **blablá** </p><p> **blabla** </p></div></body>

\*\* Body {color: red} div {color: blue} h2 {color:green} 🡪 h2 will be green the rest will be blue. H2= child.

\*\* Body {font-family; line-height} div {color: blue} h2 {color:green} 🡪 They will change FONT and LINE, but NOT colors, because font and line hasn’t been overwritten yet.

* Todo será inherited hasta que se modifique el valor especifico. NO TODO: border por ejemplo no.

**Last Rule and Specificity – Universal Selector \* :**

Pequeñas reglas de la lógica detras de CSS styling

Last Rule 🡪 Si usamos **p {color: red}** y luego **p {color: blue}.**

¡Se aplica a la última! Ganará la **BLUE**

Specificity 🡪 <p class= “red”> // .red {color: red} p {color: blue}

¡Se aplica la más específica! Ganará la **CLASS**

Universal Selector 🡪 **\*** {color: blue} p {color: red}

¡La que menos fuerza tiene! Ganará el **P**

Se usa cuando queremos resetear el Default del browser.

**Colors:**

<h3 id=”first”> **I am first** // #first **{COLOR; BGCOLOR}**

<h3 id=”second”> **I am 2nd** // #second **{background}**

* Color 🡪 Da color a textos
* BGcolor 🡪 Da color a fondos
* BG 🡪 Da color como tambien imagines **url()**

RGB (red, green, blue)

Color: rgb (00, 00, 00) 🡪 255 the max

RGBA (red, green, blue, alpha)

Color: rgba (255, 0, 0, .25) 🡪 Opacity

HSL and HEX

Color: # RR GG BB 🡪Numbers and letters a=10 f=15.

Ej: Red Color = #ff0000 🡪 F= highest, A= lowest.

Hay muchos colores que alcanza con los primeros #000 (black)

**Units:**

Absolute Values

Pixels

They control the layout of our page. 1PX = 1Dot on the screen.

H1 {font-size: 60px; width: 200px; height: 200px}

* Absolute values. They don’t change, not responsive.
* Width: largo // Height: Alto (200x200 = cuadrado)

Relative Values

Percent values

<div “outer”> <div “inner”>

.inner {color: red; width: 50%, height: 50%}

🡪 At this point nothing happens, why? Because INNER is the **child class**. We need to specify something to OUTER (**parent class**) if we are using percentages. It is **RELATIVE**

.outer {color: blue; width: 500px; height: 500px}

.inner {color: red; width: 50%, height: 50%}

🡪 Here we will have a BLUE square and inside a RED square half of the value (250x250).

EM values

1EM = 16px in default browser style, also Base Value. “Emphemeral Unit”.

<body>

<h3 “relative”> Relative

<h3 “absolute> Absolute

. relative {font-size: 2em}

.absolute {font-size: 32px}

🡪 At this point they are both the same size, why? Because default Relative is a child of Body, and base value is 1em (16px). Absolute is 32px therefore 2em.

If we change the 16px value in global… EM changes aswell.

\* {font-size: 32px}

. relative {font-size: 2em}

.absolute {font-size: 32px}

🡪 Relative will have 64px and absolute will stay at 32.

REM Values

Relative that depends on the Root. 1REM = 16px. They DO NOT depend on a parent class, they depend on ROOT values.

<div “relative>

<div “absolute”>

. relative {font-size: 2rem}

.absolute {font-size: 32px}

.html font-size : 32px}

🡪 The root would be HTML, if we change it, 2REM will become 64px.

* Using REM will help if the user changes the default browser settings (RESPONSIVE)

View Height (VH) and View Width (VW)

They are relative to the SCREEN. No solo cambian su size, sino tambien sus dimensions.

<div “banner”>

<div “header”>

.banner {width: 50vw; heigth: 50vh; color: red}

🡪 Nos dará un 50% de la pantalla, si la agrandamos se agrandará relativamente.

.header {width: 100vw; heigth: 100vh; color: blue}

🡪 Ocupara el 100% de la pantalla, sea el tamaño que sea.

Min-Height 🡪 If the content doesn’t arrive to the Minimal Height established it keeps the 100vh; if it is more content it will add more area so it fits (more than 100vh)

\*\* To adapt the area\*\*

Max-Height 🡪 Si le damos poco en comparacion al contenido deberemos usar **Overflow: Hidden//Scroll.** Para que entre.

\*\*To make sure the content doesnt surpass the área\*\*

Default Browser Styles

We can see them using Google Dev Tools (Ctrl + Shift + I), we can select Elements too (Ctrl + Shift + C).

Elements 🡪 Vemos los elements del document

“User Agent Stylesheet” 🡪 When we see this it means they are the default styles.

Style.css 🡪 We can see our own CSS file and change it, BUT it will not be saved in our file, this is just to TRY to see if we like the changes.

* DevTools are perfect for Debugging. For example Margins and Paddings

Calc() Function

This function will really help us performing math operations, also mixing and matching values.

<div “nav-bar”> This is my nav bar

<div “banner>

\* {margin: 0}

.nav-bar {bgcolor: blue; height: 100px; color: white; font-size: 3rem;}

.banner {bgcolor: red; min-height: 100vh}

🡪 Que tenemos? Una nav bar, de fondo azul, texto blanco. Y el fondo del website (banner) de color rojo. El problema es que tenemos el 100% de la pantalla + 100px del nav bar, lo que quiero es que juntos compartan el 100%.

.banner {Min-height: calc (100vh – 100px); color: red}

🡪 CALC () : Decimos que queremos la min-height de 100vh – 100px de la nav bar y deberá completar la page al 100%. DEJAR ESPACIOS ENTRE ELEMENTOS Y OPERATIONS **(**100 **–** 100**)**

* Calc() Function es lo que usaremos para completar paginas al 100% de su capacidad. DEJAR ESPACIOS ENTRE ELEMENTOS Y LA OPERACIÓN A REALIZAR. **Calc (vh – px)**

**Typography**

Properties, family, size, align, indent, etc

Font-Family 🡪 It is the typography for an element.

Font Stack 🡪 If the browser doesn’t support certain type of typography we can set up some backups fonts.

Font-Weight 🡪 Makes our fonts Bolder

Font-Style 🡪 Italic, oblique, normal (default)

* In suggestions of VSC it will appear the whole font-stack.

EJ: font-family: "Lucida Sans", "Lucida Sans Regular", "Lucida Grande", "Lucida Sans Unicode", Geneva, Verdana, sans-serif}

Google Fonts 🡪 Vamos a Fonts Google y elegimos la nuestra, luego copiamos el @IMPORT y pegamos en el CSS. Luego copiamos Font-Family y pegamos.

Text-Align 🡪 Ajusta el texto al Centro, Derecha, Izquierda (default)

Text-Indent 🡪 Sangría , podemos decir X px .

--------------------------

Line-Height 🡪 Espacio entre líneas, Relative to something (default browser)

Letter-Spacing 🡪 Space between letters

Word-Spacing 🡪 Space between words.

Text-Transform 🡪 Transformamos cada palabra del elemento elegido. (uppercase, capitalize, etc.)

Text-Decoration 🡪 Generalmente usado para Links (none, subrayado, etc.)

**CSS Box Model:**

Buscar CSS box model 🡪 Content, Padding, Border, Margin

* **El element es el TEXTO.**

Padding 🡪 Sería como expandir el **área** donde está dispuesto el texto. Paddings are internal.

{padding: A}: All Around

{padding: B B}: Top/Bot and Left/Right

**{padding: C C C C**}: Top. Right, Bot, Left.

Margin 🡪 There are default browser margins **\*{margin:0}.** Los margenes son externos al contenido (transparentes)

Border 🡪 Width Style Color **{border: 10px solid yellow}**

Border-Style 🡪 In most cases we use SOLID

Border-Width 🡪 El grosor. Ej: 10px

Border-Color 🡪 Cambia color

Border-Radius 🡪 PX. Se hacen más circulares. 50% = Ovalo.

Negative Margins 🡪 margin-top: -100px. Se solapa sobre el element superior. Podria ser útil para crear figuras complejas.

**Outline and Offset:**

El Outline se encuentra entre el borde y el margin. Utilizando offset podríamos separarla del elemento en cuestión.

{outline: Width Style Color}

{outline-offset: 10px} 🡪 Separa el “borde”, también números negativos

**Display Properties:**

<body>

<div class="block">**I am block element**</div>

<h1 class="block">**I am block element**</h1>

<p class="block">**I am block element**</p>

<span class="inline">**I am inline element**</span>

<a href="#" class="inline"> **I am inline element**</a>

<img src=".//First Project/Images/img2.jpg" width="50px" alt="" />

</body>

* Display BLOCK: Is the default. De principio a fin del renglón. They take the full width.
* Display INLINE: Es el que ocupa su espacio unicamente, como una span.

.block {bg: blue; color: White; display: inline}

.inline {bg: red; color: white; display: block}

🡪 Aqui probamos modificar sus Default properties de display

CENTRADO HORIZONTAL .6.43.48

When we learn **GRID** or **FLEXBOX,** we are will use them to center everything! Still we are going to learn some few things.

**Text-Align: Center** 🡪 Only Inline elements are gonna be centered horizontally. For the ones that are Block, only the text will be centerd.

For BLOCK elements:

1. Dar un Width
2. Dar un margin: 0 auto 🡪 Auto los centra automaticamente.

.block {width: 150px; margin: 0 auto}

Mobile Nav-Bar

We will use list-style-type

Create a ul with items:

<ul><li>Home</li><li>About</li><li>Services</li><li>Contact</li>

\* {margin :0 ; padding :0 ; box-sizing : border-box}

Body {font-family:} 🡪 This is general, how to select specifically?

* Ul li {} : Poniendo elements uno al lado del otro nosh ace ingresar en las Childs.

Ul li {list-style-type: none} 🡪 Removemos los items, numeros, etc.

ul li a {text-decoration: none; text-transform: capitalize; letter-spacing: 2px; background: #222; color: #f15025; display: block; padding: 10px 10px;} 🡪 Display block para poder ser afectada por los padding top y bottom. Sino el Browser los IGNORA, solo funciona para BLOCKS, no INLINE

**Display: Inline-Blocck** 🡪 Transforma las inline en blocks, pero mantiene su regla de Solo usa su Area, no llega hasta el final ☺

Luego será afectado por paddings y márgenes top bot.!!!!

Box-Sizing

Border-Box 🡪 Mantiene los tamaños dados y si agregamos Padding, lo mete por adentro de dichos tamaños.

.box-1, .box-2, .box-3 { width: 200px; height: 200px; color: white;}

.box-1 {background: red; padding: 20px;}

.box-2 {background: blue;}

.box-3 {background: green; padding: 20px;}

Box 1 : El tamaño es de 200x200, si agregamos padding pasaría a 200+20 x 200+20 = 240x240.

* Si agregamos Box-sizing: border-box: Mantiene el original de 200x200

None, Opacity, Visivility

<div class="none">none</div> 🡪 Display:none // NO SE VE

<div class="opacit-1">opacity-1</div> 🡪 Default opacity

<div class="opacity-5">opacity-5</div>

<div class="opacity-0">opacity-0</div> 🡪 No se ve

<div class="visibility">visibility</div>

* Opacity: Entre 0 y 1 🡪 No se ve en 0, pero existe
* Visibility: hidden 🡪 No se ve, pero existe.
* Display: None 🡪 No se ve, lo remueve incluso de las devtools

**Background Images:**

Buscamos imagines en Pixabay.

Creamos unos DIV que contengan la H1.

<div “big-img”>

<h1> **This is my BIG IMAGE**

<div “small-img”>

<h1> **This is my SMALL IMAGE**

Div {height: 300px; color: white} 🡪 Global para todos los divs

.big-img {background: url(); } 🡪 Pegamos url de la img

BIG IMG: Es más grande que el height de DIV y NO se adapta automáticamente. Si el DIV es más grande, se repite.

SMALL IMG: Es más pequeña y se repite para ocupar el height dado a DIV.

**NO REPEAT** la img ocupará su espacio original. Hay otras opciones (round, space, etc.)

**Background-Size**

COVER 🡪 Cubre todo el DIV, small imgs se distorsionan.

CONTAIN 🡪 La adapta complete al DIV; si sobra espacio NO lo cubrirá, pero mete la imagen con sizes originales.

**Background-Position**

CENTER 🡪 Al centro REAL de la DIV

Tambien hay Left, Right, and Bottom. Trabajamos con X y con Y (Porcentajes: 0%, 0% 🡪 Default)

**Background-Attachment**

FIXED 🡪 Cuando scrolleamos se mueve el texto, pero la imagen queda FIJA.

SCROLL 🡪 Default

**Linear-Gradients**

Podemos agregar unos Overlay para poder leer el texto sin problemas. Comenzemos por crear unos DIVs para quienes aplicaremos las propiedades de {Background: Linear-Gradient ()}

{L-G (**color**, color, color)} **COLORES.**

L-G (blue 20**%**, red 70**%**) % la **SOLIDEZ** del color.

L-G (**to top**, color, color, color) **DIRECCION**, Default: To Bot.

L-G (315**deg**, color, color) DIRECCION, en **DIAGONAL**

L-G (**rgba**(0, 0, 0, **0.2**), **rgba**(0,0,0, **0.9**)) **OPACITY**, de gris a negro.

**BG Image Overlay**

Objetivo: Ver el texto mas fácilmente.

**Background: linear-gradient (rgba(0,0,0,0.3), rgba(0,0,0,0.3))**

**, url(“../path.jpg”)**

Este será el background img con una overlay encima que permitirá leer claramente el texto.

\*\* Colorzilla 🡪 Online linear gradient generator!!!!! \*\*

**Positions, Float, Media Query:**

<div “banner”>

<img>

<p> lorem50

.banner {border; padding}

Img {float: left}

🡪 **FLOAT:** Elements after the floating element are gonna flow around. “Lo sacamos del flow normal”

* ¿Si queremos que el P no flote alrededor? ¿Pero si posicionar la imagen? (P comienze por debajo)
  + P {Clear: Right;} 🡪 Si float right, clear right.

Column Layout

This is just a tiny example, later we are going to use Grids or Flexbox to make it easier and perfect.

* Creamos 3 DIVs con P.
* Les damos BG Color
* Les damos Width and Height para hacerlos pequeños cuadrados

Div {height: 200px; Width: 33.33%, Float: Left} 🡪 Que pasa? Les damos un Width que sea el 100% divido 3 (3 divs), al flotarlos al left se apoyan sobre la misma línea, como si dejaran de ser block element.

\*\* Si tenemos 4 elementos : width: 25% \*\*

\*\* Elements that are following set to CLEAR: BOTH \*\*

Position Static:

It is by default. Each element it is static. If I have P2 after P1 it will be display like it. Positioned according to the normal flow.

Position Relative:

It is going to place our element relative to it’s default position. We can start using top, bottom, left and right.

Ej: {position: Relative; **top:** 200px} 🡪 Baja al element 200px de su posicion inicial.

Position Absolute:

It is going to be Relative to the parent that has position Relative. EJ: Centrar icons

<body>

<div>

<p “one”> **Lorem10**

<p “two”> **Lorem 10** <span “special”> **I am Absolute**

¿Quiénes son los parent de Special? Two, Div and Body. La “mas mayor” es BODY. Entonces si special será Absolute, el BODY será su Relative.

Ej: .special {position: absolute; top:0; left: 0;} 🡪 Esto pondrá a la special arriba a la izquierda de la página (Como relativa de BODY)

Si ponemos POSITION RELATIVE a cualquiera de los otros Parents… la Special será posicionada a partir de dicho elemento:

.two {position: relative}

.special {position: absolute; top:0; left: 0;}

🡪 Special iniciará arriba a la izquierda, PERO de la TWO

* If we use Absolute, the original space will be filled by following content.

Position Fixed:

Setting our element relative to Viewport or Screen. The space will be filled with elements.

EJ: Nav bar fixed para que se mantenga cuando scrolleamos.

.nav {position: fixed; top:0; left: 0; width:100%}

Position Sticky:

Toggles between relative and fixed, once the element is touched it will stick and become fixed. We can use Top, Left, Right, Bot.

Podemos controlar el momento en que se pega. Ej: top 20px.

A los 20px antes del elemento (por top) se pegará.

{Position: sticky; top: 20px;}

**Media Query:**

Style elements on different screen sizes. **RESPONSIVE**.

Mobile First: Comenzamos por el estilo para nuestro mobile y luego damos Querys para convertirlo a Tablet, Desktop.

@media screen and (min-width or max-width)

* Min-Width: Starting From 🡪 The moment we touch and go above the chosen PX size, it will apply changes.
* Max-Width: Up To

@media screen and (min-width: 500px) {

Body {background: blue;}

.banner {font-size: 56px;}

}

**Z-Index:**

asd

**Datos Piolas**

* DIV is for starting a new line. “**Block** level element”
* SPAN, HRefs are “**Inline** element” 🡪 To style something within a text already, so we don’t want to create a new line. (Usa unicamente el espacio necesario. Padding Margin are NOT respected. For that, I need to transform them into BLOCK)
* **Rules**: Everything will be inherited unless we change the property’s value specifically. (font-site: 1.5rem)

🡪 BORDER por ejemplo NO inherit.

* Using **REM** will help if the user changes the default browser settings (RESPONSIVE)
* **VW, VH**: Relativos a la pantalla. (RESPONSIVE)
  + Min-height: 100vh 🡪 **Full screen.**
* DevTools are perfect for **Debugging**. For example Margins and Paddings (Ctrl + Shift + I)
* **Calc()** Function es lo que usaremos para completar paginas al 100% de su capacidad. DEJAR ESPACIOS ENTRE ELEMENTOS Y LA OPERACIÓN A REALIZAR. **Calc (vh – px)**
* **CSS Box Model**: El elemento es siempre el texto y su área.
* Opacity: Entre 0 y 1 🡪 No se ve en 0, pero existe
* Visibility: hidden 🡪 No se ve, pero existe.
* Display: None 🡪 No se ve, lo remueve incluso de las devtools
* **BG: Overlay negra** 🡪 {bg: linear-g (rgba(0, 0, 0, 0.3)**,**

Rgba (0, 0, 0, 0.3))**,**

Url (“./Path.jpg”)}

* **BG URL ()** 🡪 url(“”) center/cover fixed/scroll no-repeat;
  + Podemos dar los valors pegados al URL y ahorrar espacio
* **Float:** Sacamos del “flow” normal y posiciona. **Clear:** Escapa el flow dado **(float: right 🡪 Clear: right)**
  + **Clear: Both 🡪** Si tenemos 2 floats a ambos lados.
* **Position: Relative 🡪** Mueve a los elementos desde su posición by default (inicial). Usaremos TOP, BOT, LEFT, RIGHT. The original space won’t be filled by anything. **{Position:Relative; Top: 200px}**
* **Pos: Absolute 🡪** Will be Relative to a parent (Body principamente, hasta que otro parent tenga POS:RELATIVE)

The initial space will be filled by the contenct following the element. Ej: Centrar Icons

* **Pos:Fixed** 🡪 Relative to PortView or Screen. Ej: NavBar

Ej: .nav {position: fixed; top:0; left: 0; width:100%}

* **Pos:Sticky 🡪** Relative until we touch the element, then it becomes fix. We can control the moment it becomes fixed.

Ej: {Position: sticky; top: 20px;} When it is 20px from the element It will stick.

* **@media screen and (min-width: px)** {

Body {properties: values;}

.nav {properties: values;}

}

🡪 This is RESPONSIVE: Mobile first, and use query to adapt to Tablet and Desktop.