**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:**

Does NOT work with position: static (default positioning, we should add any position). Default = 0. Para entender esto vamos a realizer lo siguiente:

* Div “Banner” // 3 imgs class “one” “two” “three”
* Damos ciertos estilos:

img {width: 150px; height: 150px; position: absolute;}

.banner {margin: 20px; width: 80vw; height: 70vw; border: 5px solid green; position: relative;} 🡪 **Relative porque nuestras imagines de Absolute, responderán a esta Parent**

.one { top: 0; left: 0;}

.two { top: 10%; left: 10%;}

.three { top: 20%; left: 20%;}

Aqui creamos que las imagines se superpongan estilo mazo de cartas. Asi podremos practicar los Z-Index.

Tenemos X (horizontal), Y (vertical) and Z (closer to me)

Queremos que ONE esté más cerca:

.one {z-index: 1} 🡪 Podemos agregar cualquier valor positivo o negativo, sea 1 o 10000

**Pesudo elements ::Before and ::After**

Create elements before and after the content, NOT Element! We must use **Content: ‘ ‘** ; NO FUNCIONA CON IMG.

P::before {content:”Hello”;} 🡪 Prints hello before the P content

P::before {content:” ”; display:block; width:100%; height: 3rem; bg: black;} 🡪 Agregamos un element block de dichas dimensiones. Sin contenido, mas que el Background Black.

Estmos Pseudoelements no figurarán en el HTML como código, sino como pseudo elements.

Expamples: Crear un borde alrededor de una imagen y disponerlo por debajo formando un mini cuadro tapado por dicha imagen.

* Usar IMG junto a un DIV.

<div “container”>

<img>

.container {width: 70vw; margin: 100px auto;}

🡪 Margin: 100px auto = Nos centra de ambos lados y nos da 100px al top y al bot.

* Queremos que la IMG ocupe todo el espacio de la Container:

Img {width: 100%; display: block} 🡪 Ocupará el espacio de block (todo el área)

* Creamos el BORDER y lo posicionamos!

.container {width: 70vw; margin: 100px auto;}

img {width: 100%; display: block}

.container::before {content:””; border: 3px solid red; width: 100%; height:100%; position: absolute} 🡪 En este punto tenemos un borde muy grande, que paso? El **ABSOLUT** responde a la **PARENT** (BODY by Default), necesitamos que sea el container con position **RELATIVE**

.container {width: 70vw; margin: 100px auto; position: relative}

.container::before {content:””; border: 3px solid red; width: 100%; height:100%; position: absolute; box-sizing: border-box; top: -30px; left: -30px; z-index: -2} 🡪 Usando Absolut + Top + Left + Inedx podemos posicionar perfectamente lo necesario.

**Selectors**

Simple: **p{} #id{} .class{} \*{}** (Universal)

Descendant: **div h1 {} 🡪** Aplica a todos los h1 dentro de Div, incluidos LI (Descendant, no son directos)

Direct Children: **div > h1 {} 🡪** Solo a los Childs directos (no LI)

::First-line: **p::first-line {} 🡪** Aplica al primer renglon

::First-Letter: **p::first-letter {} 🡪** Aplica a la primer letra del primer renglon

:HOVER 🡪 Cuando pasamos el mouse por encima sucede X cosa.

Element:hover {color: red; etc.} . Podemos agregar cualquier propiedad.

Links PseudoClasses

a:Link {color: red} 🡪 Es para los NO Visitados todavía

a:Visited {color:} 🡪 Los links Visitados

a:Active {color} 🡪 As i am clicking on it.

Root PseudoClasses

HTML element. It has the higher specificity. Generalmente lo usaremos para las **variables de CSS** y usaremos ROOT para store all of them.

:root {Font-size: }

Al tener **relatives** se modificarán **proporcionalmente al ROOT** o a la selección del usuario en el browser. **Recomendable usar REMS para las fonts.**

**Transform, Transitions and Animations:**

TRANSFORM:

Para transformar la forma de los elementos , sabiendo su Width y su Height. .div {width: ; height: }

**Transform: Translate()** 🡪 X and Y values, $, px, etc. Se mueve en relacion al tamaño de la Parent. **Movemos a los elementos en X e Y, o ambos a la vez** (20px 20px)

{Translate:transformX (50%)} {Transform: Translate (px px)}

**Transform: Scale() 🡪** Podemos aumentar tamaños X Y o ambos

{Transform: ScaleX () // ScaleY () // Scale (px, px)}

**Transform: Rotate() 🡪** Cuantos grados queremos rotar (100deg)

{Tran sform: Rotate (X deg)}

**Transform: Skew() 🡪** X or Y. Es como estirar, forma de paralelogramo.

{Tran sform: SkewX (deg) // SkewY (deg) // Skew (deg, deg)}

TRANSITION:

Para producer cambios over time, necesitamos tener algún :HOVER.

div:hover {bg: red}

.three {bg:blue; transition-property: bg; transition-duration 3s}

* Tenemos el :HOVER sobre el DIVy agregamos transition a la THREE dentro del div. Cambiará su bgcolor en 3s. **PROPERTY: El elemento que va a cambiar**

**DURATION: El tiempo que toma en cambiar**

\*\*Queremos agregar un border-radius (convertirlo en circulo)\*\*

.three:hover {border-radius: 50%}

.three {bg:blue; transition-property: bg, border-radius; transition-duration 3s, 3s}

* **Property:** Background, Border-radius
* **Duration:** 3s, 3s

🡪 Elegimos los elementos a modificar y el tiempo que lleva.

.three {transition-delay: 2s}

**DELAY: El tiempo que tarda en comenzar a cambiar**

**SHORTHAND!!!**

**{Transition: property duration delay, property durations delay, etc.}**

Transition Timing Function:

**Ease** : Default // **Linear**: Same speed start to end // **Ease-in**: Slow start // **Ease-out**: Slow end // **Ease-in-out**: Slow start, fast, slow end.

Div { width, height, bg, color, margin, Transition: all 2s ease}

Div:hover {transform: translateX(100px)}

* Hacemos que el cuadrado se mueva 100px a la derecha.

Transition: ALL 2s Ease 🡪 **ALL**: todos los elementos con hover

**2s**: Duration

**Ease:** Transition-timing-function

Probaremos uno por uno de las timing-funcitons.

ANIMATIONS:

Tenemos mas control sobre la acción, ya que Transition = de 0% a 100%. En cambio Animation = 1,2,3,4 etc.

<body>

<div class="transition">Transition</div>

<div class="animation">Animation</div>

</body>

div {width: 200px; height: 100px; color: white; margin: 10px}

.transition {background: coral; transition: all 2s linear}

.transition:hover {transform: translateX(200px)}

🡪 Rectangulo color coral, que se mueve 200px a la derecho tomando 2s en llegar. **0% a 100%**

.animation {background: crimson; **animation-name**: move; **animation-duration**: 10s; **animation-iteration-count:** 3;}

* .animation {bg: crimson; animation: move 5s infinite}
* **.animation {animation: Name Duration Count}**

**@keyframes move** { 🡪 **Animation Name : MOVE**

   0%  { transform: translateX(20px); }

   50%  { transform: translateX(100px); background: red }

  75% { transform: translateX(-200px); background: yellow}

   100% { transform: translateX(20px); background: gold}

}

🡪 ¿Que sucede? Se desplaza, at 50% cambia a rojo y sigue desplazando, at 75% vuelve atrás cambiando a amarillo, y para temrinar avanza 20px cambiando a dorado.

**@keyframes Name {} 🡪** En el medio ponemos 0%{} 100%{}. Podemos dar cualquier cantidad de % necesarios, usaremos 50% y 75%. LOS % SE APLICAN A LA DURATION TAMBIEN.

* **{Animation-Fill-Mode: Forwards}** 🡪 Luego de terminar el COUNT la animación termina y queda como el 100% declarado en @keyframes.

**CSS Variables:**

A.k.a Custom Properties 🡪 I can use the same value without “re-typing” it all over again; also, I can change globaly the variable to change some values.

**--varName: Value 🡪 How to define**

**Property: var (--varName) 🡪 How to call**

* **Variables Globales** las declaramos en **:ROOT.**

**:root {**

--primaryColor: #f15025;

--cl-seconday: green;

--mainTransition: all 2s linear;

--mainSpacing: 5px;

**}**

* Una vez declaradas procedemos a llamarlas donde sea necesario:

H1 {color: var(--primaryColor);}

H2 {color: var(--primaryColor);}

H3 {color: var(--primaryColor); transition: var(--mainTransition);}

P {letter-spacing: var(--mainSpacing); transition: var(--mainTransition);}

H3:hover {color: var(--cl-secondary);}

p:hover {color: var(--cl-secondary);}

* **Variables de Element**, las declaramos dentro de un elemento **Parent** y afectarán únicamente a sus **Children:**

Div {--primaryPurple: purple;}

.main-text {color: var(--primaryPurple);}

**Font Awesome Icons**

Buscamos iconos en Font awesome, seguimos las instrucciones y pegamos el <link> dentro del HEAD.

**<i class=” fas fas-home”>** 🡪 Tambien podemos agregar más clases a nuestro gusto

**.fa-home {color: Green; Font-Size: 10px;}** 🡪 Tambien pueden ser estilizados desde la Parent Class (div, span, etc)

**Text-Shadow:**

Se aplica al actual Text. We put some values **{X Y Blur Color}**.

H1 {text-shadow: 2px 2px 5px red;}

* Search 🡪 Text Shadow Generator

**Box-Shadow:**

Se aplica a una Box. Pasamos valores **{X Y Blur Color}**.

.box {width: 200px; height: 200px; bg:blue; margin:20px; box-shadow: 5px 0px 2px green}

🡪 Aquí creamos una caja azul con 20 margen y le damos una sombra color verde por lado derecho de 5px .

* Search 🡪 Box Shadow Generator

**Browser Prefixes:**

Especificamos el support para cada función del CSS.

Prefix for: Chrome, Mozilla, etc.

* **CanIUse .com**
* **Shouldiprefix.com**
* **We will use an extensión that will apply all these automatically.**

**Semantic Tags:**

Improves our site accessibility for screen and keyboard navigation. Reemplazarían el uso constant y masivo de DIVs. Por ejemplo: NAV, Section, Header, Footer, Article, Aside.

**Object-Fit:**

Allows us to resize images and videos to fit its containers.

<div “one”> <img> <h1> **I am a text**

<div “two”> <img> <h1> **I am a text**

* Tenemos IMGS que queremos que ocupen una linea, siendo ambas del mismo ancho y alto. Logicamente acomodaremos las Width para ocupar el 100% y agregando un espacio entre medio (width: 45%, margin: 5%) = 50%

Img {width: 100%; display: block}

.one {float: left; width: 45%; margin: 5%}

.two {float: left; width: 45%; margin: 5%}

* El width concuerda bien. El **Heigth es distinto** para ambas imágenes. Solucion:

Img {height: 300px; **object-fit:** cover;}

🡪 **Cover, Contain, Fill, None, Scale-Down**

\*\* En definitiva: Damos una **Witdh para las columnas**, Damos una **Height general para las imágenes** y le damos **Object-Fit para mostrar las imágenes** \*\*

**PRIMER PROYECTO CSS: TEA STATION**

(No Grid - No FlexBox) (We use Floats)