# first lesson-Introduction to the course:

Initial pages are usually named **Index.html**

**Tags** look like this 🡪 <html> (initial tag)

Close tags are like this --> </html>

**Titles:**

**<h1> =** Heading type 1.

**Paragraphs:**

**<p> =** Paragraph.

**HTML** develops the tags to perform a format to the web page, **CSS** improves the design and style of them. **Javascript** is useful to make dynamic information in the webpage.

**Meta UFT-8:**

**<meta x = “UTF-8”/>** = meta is to specify the sort of character that our web page can manage, where x is an attribute. The / symbol is typed to close the tag. This is another way to close an element in HTML. UTF-8 Is used to add stresses in our page.

The “meta” stands for “metadata,” which is the kind of data these tags provide – data about the data on your page.

<title> here is the value of the tag </title>



This code allows to make possible the compatibility of the code in a browser like edge.

**Responsive pages** = are those that can automatically fit their style and size according to the device: cellphone, computer, ...

# Second lesson-basic html elements :

**Titles in HTML:**

The maximum size of a title is h1, the minimum is h6.

**Paragraphs in HTML:**

It’s recommendable type any text in a tag due to the styles, if we don’t do that, we could change the style, color or any characteristic of the text. Anyway we can view a text without a tag.

<br> is for make a line break

<hr> = horizontal rule

<a> = anchor

**Links in HTML:**

**Href** stands for the webpage which we want to go.

**Target=”\_blank”** attribute is for open the url in a new window.

**NOTE:** We should use the tag’s name in lower case.

**Images in HTML:**

Just use this tag **<img>** and **src** attribute to load an image.

**Title=** it’s an attribute to show a sentence when we’re on the element.

We can upload images to the web page in a few ways: local or from another website.

**Attributes in HTML:**

We should type any attribute in HTML inside tags 🡪 <tag **HERE**> </tag>

**Alt =** alternative text, it appears when the image path is wrong.

**<pre>** is to add a new paragraph but with the space between lines. In other words, with a preestablished format.

**CSS =** Cascading Style sheets

**HTML =** Hypertext Markup Language

**Format text:**

This is the way to set a format to a text in HTML (without CSS, usually, we use style sheets to set a format to a text):

**<b></b>** make the text bold.

**<strong>** makes a text important. This could affect some processes due to the semantic of the text but are the same visually.

**<i>** set an italic font to a text.

**<em>** is to empathize in a text.

**<small>** makes a text small.

**<mark>** highlights a text with a yellow background.

**<del>** is to delete a text, it means to draw a horizontal rule in the text.

**<ins>** makes an underline in the text to stands for that the text is inserted.

**<sub>** involves a string as a subscript.

**<sup>** makes a string as a superscript.

**Character reference (charref):**

We are going to see the use of character with HTML:

**<code>** is a format used to show a certain code of any programming language.

We need to type char ref html in google search to find:

<https://dev.w3.org/html5/html-author/charref>

which shows a list of symbols to use when we cannot type a symbol from the keyboard.

**NOTE:** don’t forget the & and ; symbols to refer to any symbol in this page.

**How to add a link to send an email HTML:**

Just add the value href=“mainto:”. We use “?” to start the body message and “&” to concat another parameter.

# Third lesson-Introduction to html with css:

**Color management In CSS and HTML:**

If we type html color codes in google, we’ll find:

<https://htmlcolorcodes.com/es/>

**Hsl** = Hue, saturation and lightning

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

**Rgba**, is the color style which adds the variant a which stands for the alpha channel to set the transparency to a color.

**Border: 10 px solid rgb(…),** is a value to set a solid border.

Tabla

Descripción generada automáticamente

This picture shows how many characters are used according to a numbering system. In hexadecimal system 0 is the minimum value and F is the maximum value.

To make a color with this system we should know this syntaxis: **R** 00 **G** 00 **B** 00. It seems to the rgb system, the difference is that here we use pairs of values to set a color followed by # symbol, for example #fff is white color.

If the value of a couple is the same, we can simplify setting only one value. And If you want to add transparency to a hexadecimal color, just read this guide: <https://stackoverflow.com/questions/15852122/hex-transparency-in-colors#:~:text=The%20hex%20number%2090%20(typically,Thus%20100%25%20is%200xFF>.

**Introduction to CSS:**

Style= “Characteristic: value”.

**<style>** is used inside head tag to set a global value to a type of tag (selector, for example h1 or p).

<**link rel=”…” href=”path”**> is useful to refers to a file which we’re going to use in another file. Rel is to define the sort of relation of the file.

**Cascading in CSS:**

This name is due to the application of the styles, it applies from parent elements to derivates elements. If a element which is below to the parent has a different attribute, this one will be applied, so the attribute of the parent element won’t work on the derivate element.

**Box model in CSS:**

Diagrama

Descripción generada automáticamente

Bottom, top, right, left.

**Example**: margin: top, right, bottom, left. **WITHOUT COMMAS.**

To in horizontally the graphical elements, we’ve to use 🡪 margin: **auto** anypx;

**Text\_aling=** to aling the text in a box.

**Color’s palette:**

Just you have to see the tool’s doc which has the url from the web site which shows hexadecimal codes to improve our project.

# Fourth lesson-Links in HTML and Css:

**Status and colors of HTML and CSS links:**

To change the default status: **a:link**{}

To change the visited status: **a:visited**{}

To change the hover status: **a:hover**{} (while you’re on the cursor to the link)

To change the active status: **a:status**{} (it’s showed when you’re pressing the link)

**Text\_decoration:** is used to decorate the text, can make the text without underlines.

If you want to apply the same style to two status just type: **a:status1, a:status2**{}

**Links with images:**

just add the **<img>** tag inside **<a>** tag.

**Links as buttons:**

Just add a background color and text color LOL!!

**Relatives and absolutes routes:**

**URL =** Uniform Resource Locator.

**Absolute** path is detailed specified, instead, **relative** path isn’t.

Ip-port/name.html

Isnt’ recommendable to use absolute paths due to the ports, which could be different sometimes.

**Display:** inline\_block --> is useful to display elements which aren’t overlapped.

# FIVETH lesson-html tables:

**Tables in HTML:**

**<table>** declares a table in html, like this:

Tabla

Descripción generada automáticamente

**<tr>** = table row.

**<th>** = table head.

**<td>** = table data. To define a new line.

**Tables in HTML with CSS:**

**Border-collapse:** an attribute which collapses a border, it means that erases the space between the borders.

**Padding:** sets a space between elements.

**Border-spacing:** sets a space between borders.

**Example of a non-collapsed table:**

Diagrama

Descripción generada automáticamenteas you can see, this table has an outer border and each element has its own border, so the collapsed table is the same but with elements glued to the external border (apparently elements doesn’t have its own border in this sort of border).

**Colspan and rowspan attributes in HTML:**

**Colspan** is to set the col’s size of an element **<th>** or **<td>**

**Rowspan** is to set the row’s size of an element **<th>** or **<td>**

**CSS styles with HTML tables:**

**<caption>** is to set a title in a table.

**Line-heigh =** sets the top and bottom space.

**element:Nth-child(**even or odd**){}** 🡪 applies a property to a sequence of lines according to the parameter.

You can add a font-family from google or another website by pasting the code in <head>. Or you can see the fonts by pressing: ctrl + spacebar.

# SIXTH LESSON-LISTS IN HTML:

**HTML Lists:**

Currently, there’re three types of lists:

**<ol> ordered list**

**<ul> unordered list**

**<dl> definition list**

To add an element to a list, add **<li>** which means list item.

**List-style-type:** a value of the style attribute in a list which change the icon located at beginning of an element.

**Nested list =** a list in a list.

**Browsing menu with HTML lists and CSS:**

**Float:** a style value to order elements in a HTML list.

To apply a characteristic to a nested element, follow this syntaxis: **element nested-element**{}

For example: **li a**{} this applies properties to the **a** nested-elements in **li** elements

**Display=** displays an element in a way.

**Overflow:** hidden. Shows a list as an independent line.

**Ordered list:**

Shows the elements with a symbol which stands for the order, for example this is a ordered list:

1.one

2.two

3.three

**Type=** is an attribute which changes the symbols at beginning.

**Element::before** 🡪 is a selector which adds something before an element.

**Element::after** 🡪 is a selector which adds something after an element.

**Content:** “string”. Adds a string.

**Counter(**name**)=** a function which where name is the counter’s name

**Counter-reset: counter’s name 🡪** resets a counter.

**Counter-increment: counter’s name 🡪** increments the value of elements in a unit.

**Note:** to concat strings in html, you don’t have to add “+” symbol.

**Description list:**

**<dt>** means description term.

**<dd>** means description data.

# Seventh lesson-inline and block html elements and more topics!

**Inline and block HTML elements:**

**Inline** elements are applied in horizontal form and at the same line, and **block** elements are vertical and create new lines, like this:

Imagen que contiene Diagrama

Descripción generada automáticamente

**<span>** is an inline element to modify an element while is in a line, a new line isn’t required.

**<div>** is a block element to add a split in HTML. The split is applied on this way:

Diagrama

Descripción generada automáticamente

A div element covers all the space as if was a block, that’s why It’s called block element.

**HTML and CSS Attribute class:**

This is a way to mark a distinction between elements, if we make a attribute class, it’ll be in order

to apply styles.

This is the syntax to define a class in CSS: **.class name**{ properties which we want to apply }

To apply these properties in a HTML element just add the **class** attribute on this way:

**<element** **class=”**class name**”><element/>**

**Note:** with **\***{} you can add a property to all HTML elements.

**HTML AND CSS Id attributes:**

To define an id attribute in CSS stylesheets, follow the next syntaxis: **#name**{}

These are applied to an unique HTML element. That means you shouldn’t use it twice or more times due to a reason that will be explained later with JavaScript.

To use the defined id attribute in a HTML element, apply the class name with an **id attribute**.

Like this: **<element id=”**name**”>**

**Marks or Bookmarks with HTML:**

These are useful to add a refence to an element and drag our point of view to thereof.

This is the syntax: **<a href=”#**nameOfId**”>**

**Note:** you can combine **id** and **class** attributes in a same element.

**HTML Iframes:**

Iframe is as its name says a frame which allows to add an entire website or page to our project in the place which we’ll define.

To add a frame, just type: **<iframe src=”**reference**”>**

**Name:=”**nameOfElement**”**, is an attribute to identify an element, we could use it to load a website on a iframe for example.

# Eight lesson-html forms:

**HTML forms:**

These are displayed using the **<form>** tag.

**<Input>** is to get an input of users. Remember always add a **type** attribute.

A **Type** could be: text, submit, …

When we add an **id** attributeto a **input** element, we can later to link another element with this.

By using **for=””**. Attribute.

With **Value=””.** You can specify the info that you want to show in website.

**Placeholder=””.** Is useful for show info to clarify something in a textfield.

**<label>** is an element to add some images or text.

**To validate HTML forms:**

Use **required=”true”** attribute to validate a textfield.

The **novalidate** attribute belongs to **<form>** element which allows not to validate info.

You can also send info of a form without validate it specifying the following attribute in a submit button: **formnovalidate=”**formnovalidate**”.**

**Get and post methods:**

**(ignore the play button)**

Interfaz de usuario gráfica, Diagrama

Descripción generada automáticamente

Both of them are HTTP methods to send info from textfields, But the Post method sends query strings to server without attach it to the url as you can see in the image above, by default we use the GET method.

**Kinds of emails and numbers in HTML forms:**

Both are types of Inputs, as a password is. You also need to know that every value that you send to a server as a HTTP request is encoded, for example: blank spaces or @ are represented by + and %40.

**Radiobuttons and checkboxes in HTML forms:**

The type of **Radio** allows to display a box to select. For example, to define a gender.

**Note:** An interesting thing, is the fact that we can send two values to the server when we click on a lot of boxes, to avoid this, just name all the elements as the same, on this way you’ll send only one value. If you don’t do it, your form will see like this:

Texto

Descripción generada automáticamente

Remember to set a value in a **radiobutton**, by default the value is “on”, so if you don’t define a value this one will be sent and couldn’t make no sense with your purposes.

In the case of **checkboxes**, you can send so many values as you want. But the interesting part comes when an user clicks on two or more checkboxes, then these values will be stocked in an array to be processed by a JavaScript compiler. A check box looks like an square you know…

**Select and TextArea elements in HTML:**

To declare a **select** element just use this tag 🡪 **<select>**

This will be displayed as a box where you can select as many options as you declare.

It looks like this:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

To add options, you just need the tag 🡪 **<options>**

Make sure you’re writing this one inside a select element and don’t forget to add their corresponding values.

If you want to display an option as default, just add the **selected** attribute inside that option and assign the value of true.

With adding the **multiple** attribute in a select element you can select a lot of values in thereof. You could add the value of true to this one but isn’t mandatory.

As it says above, text area is an element so you can declare it like a tag 🡪 **<textarea>**

you can expand its size with **rows** and **cols** attributes.

**Fieldset and Legend elements in HTML forms:**

**<fieldset>** element is for decorate a HTML form and you can add the **<legend>** element to add a title in the fieldset, whose body covers all the questions as a border would do.

**Attributes of form element in HTML:**

**Action:** specifies the server where we’ll send the user’s data.

**Target:** shows the answers in a defined site, by default its value is “\_self”, which means that answers are showed at same window, but you could change it by “\_blank” if you want, for example, display the values in another window.

**Autocomplete:** determines if the browser should make suggestions for answers, its values only can be “on” or “off”.

**More values for type attributes in input elements:**

**Readonly:** defines if users can modify a textfield. Its values only can be “true” or “false”.

**Type=”**reset**”,** resets all the answers in form.

**Type=**”Hidden”, sends a hidden element to server.

**Disable:** disables an element, which won’t be changed by users and cannot be sent to server.

**Notes:** (these are more attributes to apply in CSS)

**background** attribute sets properties like functions, instead, **background-color** attributeonly sets the color.

**Border-radius:** defines the radii of the outer border, makes the border of element more round.

To add styles to elements which are inside in others with classes, follow this syntaxis:

**.class1 .class2 element**{}

If you want to modify elements with a specific type just do this: **input[type=”type name”]**{}

**Linear-gradient(color1, color2)** adds a linear gradient as color.

**Letterspacing:** In css, adds a space between the letters in an element.

**Text-transform:** uppercase, you can figure out what this means.

**Cursor:** pointer, changes the shape of the cursor when is on the element.

Diagrama, Escala de tiempo

Descripción generada automáticamente

# Nineth lesson-semantic elements in html:

**Semantic elements in HTML:**

**<section>** we can add a section.

**<article>** displays an article with this. Are elements that can be recycled.

**<aside>** adds an element in a side.

**<header> and <footer>** declares a header and a footer, respectively.

**<hgroup>** we can mix headers with this.

**<nav>** defines a navigation bar.

These elements doesn’t have a visual difference, but indeed, have a meaning, thus, the compiler can process on a different way the semantic elements, that’s why are called like this. Are useful to classify elements in CSS stylesheets and identify better the meaning of each element while we’re coding.

# Tenth lesson-CSS introduction:

**Ways to apply CSS styles:**

To make a property important and priority in a HTML element, just add “**!important**” next to the value, for example, if you have a common property in a **<style>** tag and a CSS stylesheet you can add this sentence as the below images shows:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Diagrama

Descripción generada automáticamente

This is the hierarchy of CSS stylesheets over HTML elements:

**1.** Inline CSS

**2.** Internal CSS

**3.** External CSS

**CSS classes:**

These cannot start with numbers and also can’t use blank spaces, instead, you’ve to use “-” or “\_” to denmark spaces.

To add two or more classes in a HTML element: **<element class=”**class1 class2 …**”>**

**Universal selector:**

This is the selector: **\***{}

**Subclasses:**

To define a subclass, this is the syntaxis: **element.subclass**{}, it means that a subclass is called when an element is related with it, if the element don’t have the class attached, it won’t work on such element.

# Eleventh lesson-colors, boards and more in css:

**Colors and borders in CSS:**

This is the syntaxis to specify a kind of border as property 🡪 **border:** #px **style** color;

**Note:** Here you can omit the **size** and **color** values.

You also can add this property independently 🡪 **border-style:** style1 style2 style3 style4;

**Note:** it’s not mandatory mix four styles, you could add just one or two.

**These are the border styles:**

**Dashed** borders are non-continuous.



**Solid** borders are continuous and simple.



**Double** borders are continuous and doubles.

Imagen que contiene Sitio web

Descripción generada automáticamente

**Dotted** borders are non-continuous, but with points.

Imagen que contiene Icono

Descripción generada automáticamente

There are other boards which the below image describes:

Gráfico de embudo

Descripción generada automáticamente

Groove border has its bottom enlightened, instead, ridge border is the contrary. outset border has its top border enlightened but inset border doesn’t and both have one color tone in each side.

**Note:** There’re **hidden** or **none** borders too.

**Border-width:**

It’s a value for style attribute, this one allows scale our border. You can define a value by pixels or by properties like these: **thin, medium** and **thick.**

**Border-color:** allows to set a color on a border. It follows the next syntaxis 🡪 **border-color:** color1, color2, color3, color4. It’s not mandatory mix 4 colors.

**IMPORTANT NOTE:** You can personalize a color by setting your cursor on a color like this:

Interfaz de usuario gráfica, Aplicación, PowerPoint

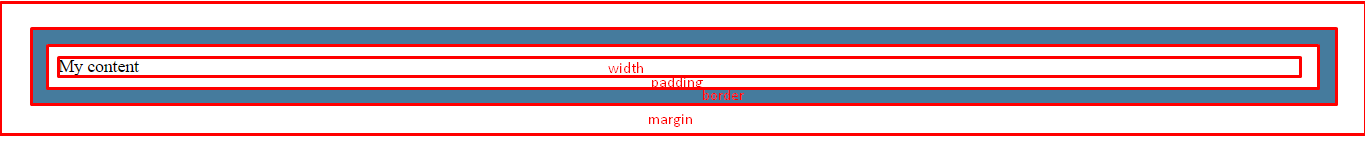
Descripción generada automáticamente

**Note 2:** from now you can check how to use classes dragging your cursor on thereof.

**Border-radius:** rounds a border according to the settled value. You can mix 4 values to round the border like this: **border-radius:** #px #px #px #px.

# Twelfth lesson-css box model:

**CSS Box model:**



**Margin** is for split our elements, **border** is for add one, **padding** is for fill the element and **content** is as it says the content.

Only a border can acquire a color.

**Width**: increases the size of **content** horizontally.

**Outline in CSS:**

Outline is located between **margin** and **border**, thus, doesn’t increase the size at all but overlaps the element. It follows the next syntaxis 🡪 **outline:** #px **style** color, where **style** is a border style.

**Outline-offset in CSS:**

Is the division between an **outline** and a **border**. You can only add a value. **Syntax 🡪 outline-offset:** #px;

**Note:** with this property you can exceed the margin of the element, then be careful with a value to this property.

**Box-sizing:** with the “border-box” value, which includes the size of **content** and **padding**, you can establish a fixed width to the box (you can modify the width with the correspond variable).

**Max-width:** With this property you can adjust the width of elements, but these will fix their size if users minimalize the window until the value stablished, which covers from **content** to **border**. If you don’t apply this, a scrollbar appears if the limit (element’s border) is reached.

**Margin property: auto:** centers an element. You can also mix this property with any value like this: **margin:** 0px auto.

**Inherit property:** inherits a value of a property from a parent element.

**Margin collapse:** it happens when two margins of different elements collapse, the greater margin will prevails over the another, that means that the element whose margin is minor would be pushed.

# Thirteenth lesson-color management in css:

I’ll omit this one cause is too obvious for readers, I explained this before xD.

# Fourteenth lesson-background management in css:

To specify the property which allows to add an image in our project, just use the following sentence:

**Background-image:** **url(**“path“**)**.

**Background-repeat:** allows to repeat an image background if zoom is less than 100%.

**Background-position:** sets a position to the image. If you reduce your zoom it can be noticed changes.

**Background-attachment:** fixed. Is useful to make a dynamic menu, with this you can allow to the background follow the scrollbar.

We can simplify all these properties except **background-attachment** with the following syntaxis:

**Background: color | url(“…”) | repeat\_value | position\_value/s |**

# Fifteenth lesson-text format in css:

Here, I’ll explain you properties that we didn’t touched before:

**Text-shadow:** sets a shadow in a text with this syntaxis 🡪 **text-shadow=** axisX axisY blur color.

**Word-spacing:** sets a space between words.

**Direction:** specifies the direction of text, by default is ltr that means “left to right”.

To remember: **text\_transforms,** **text\_aling, text\_decoration**.

**Text\_indent:** indents the text, which means that sets a space at begging of a paragraph.

**Line-height:** sets a height between lines.

**White-space:** nowrap. Allows to the text moves out of a <div> element. If you set the value “normal” you can see how the text avoid to exit from the section with line breaks.

# Sixteenth lesson-fonts management in css:

**Fonts management:**

We have a distribution of fonts types, the first we’ll take a peek is the **serif** font:

Imagen que contiene dibujo, señal

Descripción generada automáticamente

**Serif:** more elegancy and formality, **sans serif:** more minimalist and modern.

These are the most representative fonts:

Imagen que contiene Patrón de fondo

Descripción generada automáticamente

**Font-style:** by default its value is “normal”.

**Font-weight:** sets a width to a font by numbers or by properties.

**Font-variant:** helps to show the fonts in uppercase with small-caps, this means that the first letter in each word will be the greatest.

**Font-size:** sets a size to a font, values can be numbers or properties, by default its value is 16px.

**Note:** It can be used em, 1em = 16px, this unit isn’t supported in some browsers like internet explorer. To calculate the equivalency between ems and pixels 🡪 pixels/16=ems.

These are other units:

**Vh:** means view height, it changes its size if we scale the viewport.

**Vw:** it’s called view weight, it changes its size if we scale the viewport.

**Viewport** means the windows’ size. **For example**: 1vw = 1% of viewport.

**Google fonts:** you can find google font-families in internet, but if you use these by link, your project will be slower cause must load internet resources. These fonts could be imported from internet by 2 ways: by a css code, by a html link.

If you check google fonts documentation (link in tools doc) you’ll find a current version (**css2**) and more info about other characteristics which could be useful for your project, for example, font effects.

This is a simplify syntaxis to add a font:

**Font:** font-style font-variant font-weight \*font-size/line-height \*font-family;

(From now I’ll use \* to stands for a mandatory value. Elements after / couldn’t be mandatory, but you have to literally type this symbol if you want to use them).

# Seventeenth lesson-css icons:

**CDN**: content delivery network.

**Bootstrap icons:**

Here we’ll use bootstrap icons (url in tools doc), which are as its name says icons for importing to our website project, please don’t forget to check the documentation which says how to set these icons and install them.

**Google icons:**

In google fonts documentation you also can see how to use these icons, Just install these icons and type the icon’s name that you want to use in the icon tag like this:

**<I class=”…”>name</i>** (you can see the syntaxis by clicking on an seicon too (link in tools doc), I say this one due to the class because could change according to some icons).

Texto

Descripción generada automáticamente

**Ionic icons:**

It’s a framework web to make web apps (link in tools doc). You’ll see how to install it by clicking on usage button. I don’t have to explain too much here, because the documentation says everything.

# Eighteenth lesson-css display:

**Display**: it’s a property which indicates the way that elements are displayed. Some elements have by default “block”, others like <span> or <a> are inline elements so their values are “inline”.

Remember that displays an inline element means that this will stand in the same line, instead, block elements create new lines from above to bottom.

The **none** value hides the element with his box size (margin, padding, …) respectively.

If you use **visibility** property, this one hides the element but respects the space where is located.

# Nineteenth lesson-element position in css:

**Center div elements:**

It’s recommendable mix the **max-width** and **margin:** auto properties to center div elements, because with both you can center your elements independent of the device where user loads the website.

**Top, bottom, right or left** are properties of position that allow to chance the position of element.

**Static position:**

With this property you cannot move the element using properties of position (top, bottom, right or left).

**Relative position:**

Diagrama

Descripción generada automáticamente

This position takes as reference the original position to make a movement, in this example the element moved 100 px below and 50 px to the right from the position where it was.

**Fixed position:**

Fix the position relatively to the viewport. Doesn’t recognize other positions of elements.

Imagen que contiene Diagrama

Descripción generada automáticamente

Another important thing to mention is that if you use the scrollbar the element will move according to the user’s sight.

**Absolute position:**

It’s stablished according to a parent element, only recognizes **parents elements with non-static** position.

**Sticky position:**

This one won’t work in edge and to define it in opera, this is the value: **-webkit-sticky**. An element which holds a sticky position works as a **relative** element while is in viewport, when exceeds the limit of viewport, works as a **fix** element. **Note:** remember to add: top: 0 lol.

**z-index property:**

Gráfico

Descripción generada automáticamente

This position stands for the overlapping in html elements as you can see in above image. The element with the greater value will be showed first; It’s allowed elements with negative values.

To define a z-index property just follow this syntaxis: **z-index:** value (no pixels or ems, just a whole number).

**Opacity:** Defines the opacity of element. If its value were low would make a transparent element.

**Overflow:** it happens when the context’s size surpasses the container. By default, overflow is visible. If we set a value of scroll, a scrollbar will be added we could view all the content. Possible values are: **visible, hidden, scroll** and **auto**. We can also modify the x or y axes with this property summoning: **overflow-x** or **overflow-y**.

**Float:** drags elements **in a line** to the side which user chooses. If there’s no more space to place elements, then a new line is added.

**Clear**: When there’re floated elements in a line whose space is enough to add more elements yet. You can use this property to specify that you want to add the element in a new line.

**Display: inline-block:**

Allows you modify some values of box model like **margin** or **padding.** If you set the **inline** value, you’re not allowed to modify these properties.

# Twentieth lesson-css selectors:

**Decreasing css selectors:**

Once we’ve selected one element its style will be applied in decedent elements which are also called nested elements, no matter the implicit level in nested elements, css styles will be applied anyway.

You can see which elements are assigned by selectors on this way (dragging your cursor on this side, also, this image says how to stablish those selectors):

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Child selector:**

Applies styles over direct child elements only, so these elements must be in only one lower level.

To set a child selector, just add “**>”** symbol like the picture is expressing below:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

This picture also indicates that this style applies for direct child elements of <div> element in this case.

**Adjacent selector**:

Selects styles for adjacent elements which are located after the element with the styles, this image explains better what I meant.

Texto

Descripción generada automáticamente

<div> element has inside <p> element but this last isn’t the next element, so the second <p> is the right next one which will receive the styles.

To define this selector just add “**+”** symbol which stands for whose elements that are located right next to the primary element (with the styles attached).

For example:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

**Sibling selectors:**

These are which are exactly located right after the primary element. Are represented by “**~”** symbols.

For example:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**Pseudo classes:**

Are defined by **“**:**”** symbol, on this way:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

**First\_child:** applies properties to a first child element. For example:

Interfaz de usuario gráfica

Descripción generada automáticamente

In this example, the first paragraph is the first child element for the <div> element, so the font-variant: small-caps is applied properly on it.

**Pseudo elements:**

These kind of elements are represented by double dots “::”.

**Selection:** is a pseudo element which applies properties when an element is selected. For example:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

The following image shows how the paragraph looks when the cursor selects it, and the another image from above shows the properties that are applied.

**First-line:** applies properties to each element located at first line.

**First-letter:** applies an amount of properties to each element at first letter.

This example shows better the application of these concepts:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

**After** and **before** pseudo elements are used to place properties as their names says, just after or before an element.

# 21th lesson-css gradients:

**Gradients in CSS:**

**Linear-gradient(upper color, #%,bottom color, #%):** a function used in **background** attribute to make a linear gradient, it looks like this:

Forma, Rectángulo

Descripción generada automáticamente

We need to add **height: auto** in <html> element to center any element in y-axis, and **min-height: 100%** to ensure that the linear gradient expands around all the viewport. Remember: % is a ratio according to the viewport.

**Note:** you may use as many colors as you want, but remember that positions will be determined by the orientation.

To add an orientation you can use values like: **to right, to left, to bottom, to top, to bottom right, #deg, …**

**linear-gradient(to right, color , color)**

this is how a **to right** linear gradient looks:

Imagen que contiene Icono

Descripción generada automáticamente

**Radial gradients:** to define these, just do this: **radial-gradients(shape, color, color, …)** this is useful to make gradients with shapes.

**CSS shadows:**

To add them to texts just add the **text-shadow** property. This is the syntaxis:

**Text-shadow:** #px (x-axis position), #px (y-axis position), #px (blur), color.

Blur makes clearer or not the shadow.

**Box-shadow:** its syntaxis is exactly equal to **text-shadow**, is applied in boxes.

# 22th lesson-css flexbox:

**Flexbox:**

Allows to add flexibly and effectively elements, to add flex boxes just add the value **flex** in **display** property.

**Flex-direction:** allows to order in a way elements which are placed in flex boxes. Its values are: **column, column-reverse, row** and **row-reverse**.

**Flex-wrap:** allows to wraps or not elements according to its value: **nowrap, wrap, wrap-reverse.**

**Flex-flow:** with this you can mix **flex-wrap** and **flex-direction** properties. **syntaxis:** direction | wrap.

**Justify-content:** allows to justify in a way the content in a flex box. There’re a lot of values for this property, so I recommend to see the documentation in google.

**Align-items:** this property is useful to align elements horizontally, by default its value is **stretch**.

**Baseline** is to place elements according to an horizontal rule in middle of elements.

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

**Flex-start:** places items at top according to a horizontal rule placed a top of elements.

**flex-end:** places items at bottom according to a horizontal rule placed a bottom of elements.

**Center:** centers elements. **Note:** we can notice better any change modifying font-size or height of elements.

**Aling-content:** aligns elements vertically, so we work with vertical lines which are models for place elements here.

**Order:** is useful to place the order of elements in a flex box. But it’s recommendable apply this property to each element to avoid problems, in this way:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media

By default each element has an **order** value equals to zero.

**Flex-grow:** changes proportionally the size of elements.

**Flex-basis:** it’s the same as flex-grow, but when we reduce the viewport’s size, the element’s size prevails.

**Aling-self:** Aligns only the element with this property. So, you must apply it individually.

# 23rd lesson-JavaScript introduction:

Javascript is a programming language interpreted by browsers. Javascript is used in frontend (what a client looks) or backend (server side). To work in backend we’ll use Node.js framework, if you want to search for other frameworks for frontend working, it’s recommendable to use React, Vue.js or Angular.js. JavaScript is based on ECMAScript standard.

# 24th lesson-JavaScript data types:

There are some data types in JavaScript: **string, number, object, Boolean, function, symbol, undefined.**

These are dynamic, in other words, we can reassign every time we want any value.

**TypeOf:** help us to check the data type of a variable. To summon it just write: **typeof** variable.

To define a symbol just type: **Symbol();**

**Classes** are also **function** data types. **Undefined** type is just a variable without any value assigned.

**Null** is an object. **Arrays** are objects too. **Empty strings** are strings data types.

**Concat strings with numbers:**

It’s possible to concat numbers with strings, but you have to know these rules.

1. If you type at beginning numbers, the operation will be numeric. But after any string, numbers will be treated as strings data types:

Texto

Descripción generada automáticamente

1. If you concat numbers after any string, you can make numeric operations just by closing them in parenthesis.

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

**Var, let and cons:**

Values in JavaScript are also called **literals.** You can define a variable type of Var just by writing its name, but this is not a good practice, it’s recommendable to add at beginning the variable type whether **var, let** or **const**.

Remember: **const** variablescannot be redeclared and reassigned, **let** cannot be redeclared but allows a reassignment, **var** lets you do both stuff.

**Good practices for variables names:**

You should apply the camelCase style, I mean, typing variables like this: **nameOfVariable, wellDone, …**

It’s recommendable to let space between parenthesis for a better comprehension of your code.

**Function ( “here” ).**

Variables can be defined in this way too:

Texto

Descripción generada automáticamente

But remember that JavaScript compiler take in count uppercase or lowercase letters. So, you must be careful with the exactly name of variables, also, you cannot initialize a variable with numbers.

Tabla

Descripción generada automáticamente

This is a list with reserved words which aren’t recommendable to use in our variables names.

**JavaScript operators:**

Those are the JavaScript operators:

Tabla

Descripción generada automáticamente

This is de difference between post-increment and pre-increment in a nutshell.

Texto

Descripción generada automáticamente

Where a=1 and b=2.This applies for decrement too. When you use post-increment the variable z gets the original value of a and a increments in a unit, but in pre-increment, first the value of b increments in a unit and then is assigned to the variable z, that explains why both have different values.

**Hierarchy of operators:**

The following table shows which operators are more important and its associativity:

Tabla

Descripción generada automáticamente

**Assignment operators** are preceded by an operator followed by = symbol.

For example: +=, -=, \*=, …

**Comparison operators** are these: =, ==, ===, !=, !==

= is for assignments, == compares values only, === to compare values and data types.

For example: 2 == “2” is true, but 2 === “2” is false.

!= and !== are the contrary of the previous operators.

**Relational operators** are <, >, <= and >=;

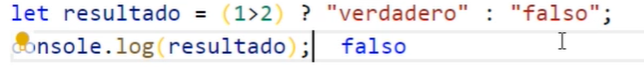
**Booleans operators** are && and ||, I don’t have to explain a lot in this.

**Ternary operators:** follows this syntaxis:

Diagrama

Descripción generada automáticamente

For example:



**Number()** converts a string to number.

**IsNaN()** checks if a value is a number.

# 25th lesson-control sentences:

**If/else sentences:**

They are conditions to apply on our source code, I won’t explain this cause it’s too easy lol. What you’ve to know is that it’s a good practice including else if sentence instead if.

**Switch sentence:**

Unlike if/else sentences, these uses expressions with string or number values associated which are useful to determine which sentence must be executed. This is the syntax of switch sentence:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

It’s recommendable to add **break** sentence cause if we don’t, code will continue in execution. I mean, other cases will be evaluated by the compiler. As **else** sentence works, **default** sentence is executed when the JavaScript compiler cannot find a case to work.

You can also gather a lot of values in a case, for example:

Texto

Descripción generada automáticamente con confianza media

**Switch** uses strict comparisons with values and data types, be careful to not confuse a string with a number.

# 26th lesson-JavaScript loops:

**While:**

Executes sentences while the condition is true.

**Do while loop:**

Unlike **while** loop, executes at least one time the content between curly brackets {}, it follows this syntax:

Imagen que contiene Texto

Descripción generada automáticamente

**For loop:**

With this loop it’s mandatory to define **variables, a condition and increments/decrements.**

**Continue** sentence is for go to the next iteration without executing the rest of the loop.

**Labels**:

They’re denoted like this: **label:**. Are useful to indicate to which part of our program we want to go. This is the way to declare a label:

Texto

Descripción generada automáticamente

The label must be placed afterwards a **continue** or **break** sentence, the effect of them will be executed before to refers the label. This is a style of programming called go-to, but isn’t recommendable for our programs.

# 27th Lesson-JavaScript arrays:

You could declare an array with **new array (…);** but isn’t recommendable due to it’s a deprecated way to do it, you don’t need to define the array length to declare it.

An alternative (but not recommendable) way to push elements is this:

Texto

Descripción generada automáticamente

In this way, we could add some holes between arrays, so we could get some problems with our projects if this happens, like memory leaks.



ECMAScript version 2015 added this function to check if an object is an array. **Array.isArray(object);** Where **Array** is a class, **isArray** a method and **object** the parameter that we want to check.

# 28th lesson-JavaScript function:

Functions are reusable code, I think you only must know this:

Texto

Descripción generada automáticamente

Where **return** sentence allows you to get a value from the function. This is an example called self.

These are expression functions:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Anonym function doesn’t have a name as you can deduce.



But isn’t recommendable to use them cause you can only summon one of these just one time.

A function is an object, so is possible to use their methods (inside the function), like **arguments.length**.

Texto

Descripción generada automáticamente

These are arrows functions, which don’t require **return** sentences or curly brackets “{}”.

Texto

Descripción generada automáticamente

It’s matter to make something clear. Cause a **parameter** isnt’ an **argument**, parameters are a list of variables defined by users at definition of the function, **arguments** are those we set when a function is summoned and overwrite **parameters**.

These are some examples where you can notice better any difference.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

As you can see, it’s not required an argument to summon your function, **arguments[x]** has the arguments that I’ve sent.

**Return** sentences are added explicit by users, but if they don’t want to add it, it’s just settled by the compiler by default

**Pass by reference:**

When we define an object, this has a memory reference, for example 0x333, and if we summon a function with this object as an argument or parameter, said argument or parameter would point to the object, so in other words, we’re passing the object with a reference.

# 29th lesson-JavaScript Objects:

An Object can get **properties** and **methods,** each one stores a memory reference. **This** sentence allows to access to any property while you’re in an object. For example:

Texto

Descripción generada automáticamente

Here you can see how to summon a method of a function and use the **this** sentence, which requires the name of a property or method that belongs to the object. **REMEMBER:** you must add parenthesis when you want to summon a method because are functions.

This is another syntax to make an object:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media

You can also get a property of an object with this syntax (using brackets):



Or to make a “**for”** loop with “**in”** sentence to simplify the code:

Imagen que contiene Texto

Descripción generada automáticamente

If you want to delete a property, just use **“delete”** sentence:

Pantalla de un video juego

Descripción generada automáticamente con confianza media

**Methods to display** **an object:**

**Object.values( object’s name )** is useful to return an array of properties and methods which belong to an object.

**JSON.stringify( object’s name )** allows to change an object to string.

**Get and set methods:**

**“Get”** allows to set a function in an object, but to summon it, you must omit the parenthesis:

Texto

Descripción generada automáticamente

**“Set”** is almost the same case, but you can set a value to a property with this method:

Texto

Descripción generada automáticamente

**Constructor methods:**

Allows to build objects with properties and methods, it’s recommendable to set a constructor’s name in upper case. For this purpose, you must use the function notion and the **“this”** sentence:

Captura de pantalla de computadora

Descripción generada automáticamente

For adding a new property or method to all objects without modifying the constructor, you need the **prototype** method which is applied between the constructor and the new property in this way:

Captura de pantalla de computadora

Descripción generada automáticamente

**Call** method allows you to call a missing **method** of an object with another object with a similar structure, in other words, invokes a method from an object using the data of the another:

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente

If you set parameters to the function, this is the way to give arguments when you summon **“call”** method:

Captura de pantalla de un celular

Descripción generada automáticamente

**“Apply”** method is the same as “call”, but requires an array of values when additional arguments are required:



# 30th lesson-JavaScript Classes:

Unlike JavaScript constructors for objects, classes follow this syntax:

Captura de pantalla con la imagen de una pantalla

Descripción generada automáticamente con confianza media

The difference as you can see is that a class must use the **“class”** sentence and a constructor.

**“set”** and **“get”** methods use the same syntax as constructors of objects do. But it’s recommendable to apply “\_” to name any property in order to make easier the job of name these methods.

We can, due to **hoisting**, to summon a defined function after its invocation, but we cannot to build an instance of a class before thereof.

**JavaScript inheritance:**

To apply inheritance to a class, which means to make a child class with same properties and methods that belongs to a parent class, we need to use “**extends”** sentence and to summon the super constructor (parent constructor) in this way:

Texto

Descripción generada automáticamente

**Overwrite methods:** If we overwrite a method in a child class, this one will be used instead of the parent method. But this is useful if you use **super** sentence to summon the parent method’s body.

Texto

Descripción generada automáticamente

**Object class:** Is the parent class of any class in JavaScript. We can apply polymorphism with this class, which means to use different behaviors of methods according to the definition in each class. For example:

Texto

Descripción generada automáticamente

Una captura de pantalla de un celular

Descripción generada automáticamente con confianza media

As you can see, each instance applies the defined method of its class.

# 31th lesson-static sentence:

**“static**” summons a function using the class name, you cannot use these methods with an object, for example:

This is de definition of a static function:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

This is the invocation:



If you noticed, the way to summon a static function is without console.log() method, since those functions are associated with classes, not with objects. So, you can only view the output in the console.

**Static JavaScript attributes:**

how static methods to work, these also require the class name to get access them: Texto

Descripción generada automáticamente

In this example, the variable “howMuchPeople” increases when a new object of class Person is built. These attributes can be inherited too. **Static** attributes are useful, for example, to make ids for people or objects with the purpose to identify them.

With this brainstorm, you could deduce something else: non-static attributes only can be associated to objects. So this is the way to define them:

Imagen que contiene Escala de tiempo

Descripción generada automáticamente

for make constant static attributes, this is the way, they are useful, for example, to define a specific limit of population:



# 32th Lesson-JavaScript strict mode:

To add this mode just type: **“use strict”** in the first line of your source code. Can be used in a function but it’s recommendable to write this sentence at beginning of your project. With this mode you cannot use variables or functions which aren’t declared before.

# 33th lesson-JavaScript overwrite:

This phenomenon occurs when you literally overwrite a function in a child class. It’s possible to summon the function which we overwrite using **super** sentence. **Polymorphism** can be executed when we use a function which executes the same function’s name of two or more objects. **InstanceOf** is a method that allows us to check if an object is an instance of a class. Uses this syntax: **object instanceOf class;**

An advice for you, is to write first the classes with less hierarchy, at the end you should paste the classes with the greatest hierarchy.

# 34th Lesson-Errors Management:

**Try{}** and **catch( parameter ){}** must be used when we’re gonna to execute some fragment of our source code which could contain an error for throw an exception message. **Try** is for execute the code with possible errors, **catch** is whose catches errors and display the exception messages. We can add at the end **finally{}** method to execute something after the analysis of errors, it’s not mandatory. **Throw** sentences are used to send exceptions in console, usually, for validations.

A pre-defined error message has a **name** and a **message**:

Texto

Descripción generada automáticamente con confianza media

# 35th lesson-Arrow functions:

We talked about this topic before, but it’s important to mention that the **hoisting** notion, which means to summon a post-defined function, isn’t applied with these kinds of functions.

These are some ways to define arrow functions:

Imagen de la pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

# 36th lesson-Callback function:

A callback function is that which is used as argument in another function. **Asynchronous callbacks** are those which they run in a sequence according to the time settled in them, for this purpose we use **setTimeout()** method in this way:

Texto

Descripción generada automáticamente

This is in a nutshell what I said:

Diagrama

Descripción generada automáticamente

There’s another way to make an asynchronous callback using **setInterval(),** which executes infinitely at a defined time interval a function:

Texto

Descripción generada automáticamente

# 37th lesson-JavaScript promises:

Diagrama

Descripción generada automáticamente

Promises are basically executed code; it works as if/else sentences do. You can define promises in this way:

Texto

Descripción generada automáticamente

As you can see, you can use **then** method with the two cases, but also you can add both methods using separately these cases as arguments. **Value** and **error** variables are arguments given by the **resolved** and **rejected** methods respectively, in other words, the cases.

**Note:** if you add **async** sentence before a function, you’re gonna indicate to compiler that you’re making a promise:

Texto

Descripción generada automáticamente

We can process the promise without **.then()** using **await** sentence:

Texto

Descripción generada automáticamente

**Note: await** only can be used in a function declared as **async** and freezes the execution for a while, I mean, you cannot execute code after an **await** sentence if it doesn’t end the execution.

# 38th lesson-HTML Dom management with JavaScript:

Diagrama

Descripción generada automáticamente

This is a model of a HTML document. To incorporate JavaScript code in one of those documents you must add **<script>** element at the end of any other html element, since if you don’t, the JavaScript compiler won’t catch and process these html elements.

To get access to any html element, define an id for the element which you want to use and just follow this syntax: **document.getElementById(‘idOfElement’).**

If you want to get the text in these elements add at the end this method: **.innerHTML.**

It’s possible to get access to an array of html elements by tag names with this method: **.getElementsByTagName(‘tag’); Note:** remember that a tag name is the name of an element.

Indeed, you also can return an array of html element by classes: **.getElementsByClassName(‘class’).**

🡪 **.querySelectorAll():** with this method we select some html elements which also have a specific CSS style.

🡪 **.forms[‘id’]:** a method of “document” object, allows to get access to a html form element using a form id, if you want to access individually to each element add “**[‘nameOfElement’]**” at the end. Don’t forget the sentence: **.value** to get access them**.**

**Onclick:** it’s an attribute of an html button, useful to execute JavaScript functions when users click on it.

**.write():** an method which belongs to “document” object. Overwrites all the website when is summoned after the document loads, everything literally.

**Note: style** attribute contains values as we know, to access them just type: .value, for example:

**(…).style.color;**

# 39th lesson-Events Management with DOM and JavaScript:

To search an event, you can type the prefix “**on**” and you’ll view many options to choose:

Texto

Descripción generada automáticamente  
when it’s used an event in a HTML element, a “**this**” object is created to point at element, for example, we can use the mentioned object for set a new inner value in a tag:



If you wanna do the same thing but with JavaScript, write the following code: **document.getDocumentById(‘the ID’).onclick = set here the event**;

**Note:** If the event is a function, don’t add parenthesis at the end.

**Onload event:** displays an event when the website is immediately loaded.

**Alert():** Is useful to display a pop up screen when we load the website.

**Navigator:** a pre-defined object when we work with websites in HTML.

**.cookiesEnabled:** an method of Navigator to check is cookies are enabled.

**Onchange event:** Is executed when an user changes the value of an element, is often applied in text fields.

**Onmouseout and onmouseover events:** both are executed on an element when the user drags the cursor on or out of element respectively.

**Onmousedown and onmouseup events:** both are executed when we click on or release an element.

**Onfocus and onblur events:** Are executed when the cursor is focused or lost its focus on an element, are often applied in text fields.

**.addEventListener():** adds an event listener to an element, it follow this syntax: **( ‘event’, method to execute, boolean )**; **Note:** the event name doesn’t contain at beginning the prefix “**on**”.

**Note2:** It’s allowed to use arrow functions as methods to execute in **.addEventListener().**

**Note3:** The Boolean value is assigned with the purpose to declare if is necessary for you to apply the event to all child element in the father element. It’s useful to apply styles in forms.

**Element.target:** it works as **“this”** pointer does, with **“target”** method you can get access to the HTML element. This sentence is useful when we need to chance any style through events in JavaScript.

**.classlist:** a method which allows to return all style classes applied in an element (returned from document object).

**.toggle(‘class name’):** It’s a method which allows to remove or add style classes temporarily in an element.

# Additional content:

* <https://www.youtube.com/watch?v=qphme6ULtq0> (How to make image animation with HTML and CSS)
* <https://www.youtube.com/watch?v=9WChhTf5mPc> (to remove backgrounds in images)
* <https://www.w3schools.com/css/css3_2dtransforms.asp> (transforms methods)
* <https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseenter_event> (mouseenter event)
* <https://w3c.github.io/uievents/#event-type-mouseenter> (mouse event types)
* <https://www.youtube.com/watch?v=ppcBIHv_ZPs> (multiplayer snake game)
* <https://www.youtube.com/watch?v=M9RDYkFs-EQ> (html snake multiplayer game deployment in haroku)