Basic structure and tabs

Forms

CSS

Position property

# PREREQUESITES

<https://marketplace.visualstudio.com/items?itemName=ms-vscode.live-server>

# A screenshot of a computer Description automatically generated

# Basic structure and tabs

|  |
| --- |
| <!DOCTYPE html> <!—Indicate that this file is a HTML file-->  <html lang="en"> <!—Indicate the lenguage of the html page-->  <head><!—In the Head we describe some global properties of the web page like:-->  <meta charset="UTF-8" /><!— type of encoding for the characters to be used -->  <title>Page Name</title><!—The name to show in the browser tab -->  <link rel="stylesheet" href="style.css" /><!--indicate where is the external style file -->  <script src="scripts.js"></script><!--indicate where is the external javascript file that dynamise the web page  </head><!— End web description -->  <body> <!— content visible on the website -->  <header><!—First visible part of the web page -->  <h1>Title 1</h1>  <nav><!--🡪<!—Index to navegate thought the web this page or other extenal links -->  <a href="/">INIDEX</a><!—links-->  <a href="/articles">ARTICLE</a>  </nav><!—End Index-->  </header><!—End first part of the web page-->  <main><!—we write here the content of the website -->  <section><!—is used to divide the content into thematic. Unlike <div> , all the content inside <section><section/> is related-->  <article><!—Independent content -->  <h2>Títle2</h2>  <p>Paragraph.</p>  </article><!— -->  </section><!—End related content -->  </main><!—End webside content -->  <aside>relational information </aside><!—Use to put additional elemnt , like anouncementes or a sidebar -->  <footer>Copyrigh © 2024</footer><!—Last visible part of the web page -->  </body><!—End visible part of the web page -->  </html><!—End html document--> |
|  |

# Forms

The tag <form action=” url where to send the data collected by the form\*” method=”get” > wraps the form elements. It specifies the action attribute, which indicates the URL where the data will be sent, and the method attribute, which specifies the HTTP method to use (“**get”** sends the information in the URL visibly and “**post”** sends the information in the body of the HTML request). inside the form we have pairs (label(optional),input) of text <label for="X"> and space to enter information<inputs id="X"> related to each other by the "for" and “name” attributes.

|  |
| --- |
| <form action="" method="get" class="example">  <div class="example">  <label for="formName">Enter your name:</label>  <input type="text" id="nombre" name="forName" required />  </div>  <div class="example">  <label for="email">Enter your email:</label>  <input type="email" id="email" name="email" required />  </div>  <div class="example">  <input type="submit" value="¡Subscribe!" />  </div>  </form> |

At the end of the form we have a button to send the information <input type=”submit”>.

There is many tipes of input elements:

<https://developer.mozilla.org/es/docs/Web/HTML/Element/input>

|  |
| --- |
|  |

# CSS

There are three methods to apply styles:

Inline:

|  |
| --- |
| <p style=”color: red;”> text to change it color</p> |

In a section within the header

|  |
| --- |
| <head>  <style>  p{  color: green;  }  </style>  </head>  <body>  <p >text to change it color</p>  </body> |

And in an external file.

|  |
| --- |
| <!--index.html- - >  <head>  </head>  <body>  <p >text to change it color</p>  </body> |

|  |
| --- |
| /\*style.css\*/  p{  color: orange;  } |

When we use internal and external styles , we use a “selector” to indicate the element we want to style. In the example above to refer to all <p> elements on the web page.

The **inline styles have higher priority** than internal and external styles. If we want to prioritise a style we have to use the word !important.

|  |
| --- |
| <!--index.html- - >  <head>  </head>  <body>  <p style=”color: red;”>text to change it color</p>  </body> |

|  |
| --- |
| /\*style.css\*/  p{  color: orange !important.;  } |

It is necessary to understand that there are two types of html elements according to their behaviour when stacked next to each other on the web page.

Block elements like: <p><h><div>,they fill the entire width of the web and generate a line break. If we inficate in the style a specific width and height it respect it.

Line elements like:<span>. They fill only the width necessary for their content, and don’t generate a line break, so more than one can appear on the same line. If we inficate in the style a specific width and height to this element , it doesn’t do anything.

we can add the css property dysplay to modifay this behavior:

“dysplay: inline” doesn’t respect the specified width and hight, the element uses a minimun size for its content and doesn’t generate a line break.

“dysplay: inline-block” respect the specified width and height, but doesn’t generate a line break.

“dysplay: block” respect the specified width and height, and generate a line break.

Every elemnt independenly of the type is incampulate in a box with the next componentes.

A diagram of a variety of sizes

Description automatically generated with medium confidenceA green and pink square with black lines

Description automatically generated

we can define the : with, style and color for the border, see (https://www.w3schools.com/css/css\_border.asp). But for the rest this options are not avalible.

By default some brosers use the property **contet-box** that asume that the sice of the element is thes sice of the content more the sice of the padding and border. It is recommendable change it in the way that for the element only take care of the content. To do this we use :

|  |
| --- |
| \* {  box-sizing: border-box;  } |
|  |

This guarantees the same behaviour in all browsers.

## Position property

The elements in a web page start to be painted from the top left corner of it container.

If we want to change it position using the properties:

X -axi: **left** and **right** .

Y- axi: **top** and **bottom**.

Z-axi: **z-index(**indicate, in the case of overlapping of two elements, which is on top, the one with the bigest z-index value).

First we have to change the "position" property which by default has the value "static" to one of the following:

### relative

It allow move an element with respect to its position in the normal flow of the web page.

### Absolute

The element will move relative to the to its primary container which is usually the entire website.

### fixed

The element is positioned in a fixed position inside it container, maintaining it even when scrolling through the web page.

### sticky

The element will have the same behavior that a relative element, unless it goes out of view. Then the element will behave as if it were a fixed element.

\*The container can be another element, like in the next example where the div with class “relative” contain two other divs, one with class “absolute” and other with class “sticky”:

A screenshot of a computer

Description automatically generated

|  |
| --- |
| <div class="relative">  <h2>Relative</h2>  <p>  In order for a secondary element to have an absolute position, the element must be relatively positioned.  </p>  <div class="absolute">  <h2>Absolute</h2>  <p>  An absolutely positioned element is positioned in relation to its main container.  </p>  </div>  <div class="sticky">  <p>I will not move until my container finised.</p>  </div>  <p>  some text inside the relative div object.<br>  to can do scroll.<br>  and text how the sticky property works.<br>  when we do scroll the sticky text doesn't move<br>  and like the fixed tex has a lower z-index<br>  it pass down the sticky text but over the absolute one that doesn't have any defined z-index.<br>  </p>  </div>  <div class="static">  <h2>Other element</h2>  <p>this has a static posicion, it is by defaul</p>  </div>  <div class="fixed">  <h2>Fixed</h2>  <p>  It is positioned relative to the browser window,  so it always stays in the same place. Like relative position,  it also uses the top, bottom, left and right properties to specify its location.  </p>  </div>  ---------style.css---------  .relative {  position: relative;  border: 3px solid blue;  }  .sticky {  position: sticky;  top: 0px;  left: 0px;  background-color: purple;  z-index: 2;  }  .absolute {  position: absolute;  right: 0;  background-color: orange;  }  .fixed {  position: fixed;  bottom: 0px;  right: 0px;  max-width: 25rem;  background-color: yellow;  z-index: 1;  } |

## Inheritance and CSS variables

In a web page to give omogenity, we use inheritance and custom properties.

**Inheritance**

An element that contains another element, e.g. a paragraph inside the body inherits the css of the parent element, i.e. if we define the font for the body, all elements inside the body will by default use that font.

**Custom properties**

Normally we use “root” and “var” in css to define a range of colours to be used on the web page, for color palette we can use : [https://mycolor.space/.](https://mycolor.space" \t "_blank)

Furthermore, the “:root”’s elements can be modified using Javascript.

A screenshot of a computer

Description automatically generated

## Display:Flex and GRID

These two properties allow us to arrange the way in which each div element within a parent div element is displayed on the screen; the elements will automatically adjust to the screen according to the instructions we give them.

One of the main differences is the flex-wrap option, which does not exist in grip and allows the elements to move as the screen size changes. Another difference is that grip allows us to select element by element using the same class, but with flex we have to define a different class for each element if we want to change the css for one of them in a different way to the rest of the group.

### Flex

A good page to check what happen when we change the properties asociated to display:flex , that are:

flex-direction : row o row-reverse o column o column-reverse

flex-wrap: wrap or nowarp

/\* alignment for horizontal axis

justify-content: flex-start o flex-end o center o space-between o space-arround;

/\*alignment for vertical axis

align-items: flex-end o flex-start o center;

is : <https://codepen.io/enxaneta/full/adLPwv/>,

but the best way is to build our own simple website for testing.

<!DOCTYPE html>

<html lang="en">

<head>

<meta chartset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

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

<title>Intro Flexbox</title>

</head>

<body>

<div class="container">

<div class="box box1">1</div>

<div class="box box2">2</div>

<div class="box box3">3</div>

<div class="box box4">4</div>

<div class="box box5">5</div>

</div>

</body>

</html>

A screenshot of a computer screen

Description automatically generated

### Grid

[https://grid.layoutit.com/ --> generate](https://grid.layoutit.com/%20--%3e%20generate) code and show how it looks

Let us organice the espace distributing it in a equitatibe way. We can combine the use of grip and flex.

<!DOCTYPE html>

<html lang="en">

<head>

<meta chartset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

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

<title>Intro Flexbox</title>

</head>

<body>

<div class="container">

<div class="box">1</div>

<div class="box">2</div>

<div class="box">3</div>

<div class="box">4</div>

<div class="box">5</div>

</div>

</body>

</html>

A screenshot of a computer

Description automatically generated

Instead of use grid-template-rows or grid-template-columns we can use areas.

A screenshot of a cell phone

Description automatically generated

:root {

--bkcolor: cyan;

}

.container{

display: grid;

/\*grid-template-columns: 1fr 1fr 1fr;

we can write-columns: repeat(3,1fr);

grid-template-rows: 1fr 3fr 1fr;

grid-template-columns: 1fr 3fr 1fr;\*/

grid-template-areas:

"header header header header"

"sidebar main main main"

"footer footer footer footer";

gap : 20px;

}

.box {

background-color: var(--bkcolor);

padding: 20px;

font-size: 30px;

text-align: center;

}

.box:nth-child(1) {

background-color: pink;

grid-area: header;

}

.box:nth-child(2) {

/\*with order we can move element inside the grid‰\*/

grid-area: sidebar;

}

.box:nth-child(3) {

grid-area: main;

background-color: red;

}

.box:nth-child(4) {

grid-area: footer;

background-color: violet;

}

### Grid and divs

A grid divs can contain flex divs and a flex div can contain grid divs

<!DOCTYPE html>

<html lang="en">

<head>

<meta chartset="UTF-8" />

<meta http-equiv="X-UA-Compatible" content="IE=edge" />

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

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

<title>Intro Flexbox</title>

</head>

<body>

<div class="container">

<div class="box">

<div class="flx">1</div>

<div class="flx">2</div>

<div class="flx">3</div>

</div>

<div class="box">

<div class="flx">4</div>

<div class="flx">5</div>

<div class="flx">6</div>

</div>

<div class="box">

<div class="flx">5</div>

<div class="flx">8</div>

<div class="flx">9</div>

</div>

<div class="box">

<div class="flx">10</div>

<div class="flx">11</div>

<div class="flx">12</div>

</div>

</div>

</body>

</html>

A screenshot of a computer

Description automatically generated