

CSS

CSS Recap

- Cascading Style Sheet - (Cascading - DOM children inherit CSS properties)
- Used to write style **rules** and determine the look or appearance of a website.

Using CSS

- To use CSS we create a file with the **.css** extension
- Add it to the <head> tag of an html document
- And link the CSS with the HTML like so:

```
<link rel="stylesheet" href="file.css"/>
```

F12 is helpful when writing CSS too as we can see styles applied to the elements in the **elements** 😊 There you can see styles that have been applied and may that have not due to some writing error.

Basic syntax

Think of CSS as rules which are defined for **tags, classes, ids and others** (more in CSS Selector Types part). The rule has a selector which determines to which elements it should apply a style. Inside there are properties which have a key and a value, for example:

```
div {  
  width: 100px;  
}
```

In this example the selector is for all **div** elements and it sets a property of key **width** to value **100px**. This structure is used throughout all CSS.

CSS Selector Types

CSS Selectors are style rules that are applied. There are a few types of them.

Most important:

- **Element selectors** - selects the elements/tags in a HTML document.

```
div {  
  width: 100px;  
}
```

- **ID selectors** - applies the rule on a specific ID. Only works on HTML elements which have that specific id assigned with **attribute** **id="something"** where something is your id name 😊

```
#something {  
  color:red  
}
```

- **Class selectors** - applies the rule on a specific class. The same as **ID** but classes can/should be used on multiple elements.

```
.container {  
  color: blue;  
}
```

- **Pseudo-class selectors** - applies style to specific pseudoclass. There are many of them. But most important could be:
 - hover
 - focus
 - first-child
 - nth-child(n)

Example:

```
button:hover {  
  cursor: pointer;  
}
```

This style will be applied to all button elements who have a **hover** pseudoclass. When does the pseudoclass get applied though? Usually they are internal classes, for example, the **hover** class is applied to an element when a mouse cursor is positioned over the element (a developer does not have to apply it manually, it is done by default). The same with pseudoclass **focus** - it is applied when a input element is clicked or "focused".

Combining selectors → there are 2 ways to combine selectors.

- With space. This combination will be applied to all children who satisfy the rule. **Example:**

```
.container .date {  
  color: blue;  
}
```

This selector will be applied to all elements with class **date** which are nested inside class **container**. For example:

```
<div class="container">  
  <p class="date"> 2020-12-04 </p>  
</div>
```

The color blue will be applied to `<p>` tag as it is inside class **container** and has class **date**. Also, this will get applied to elements even if they are nested inside others... Example:

```
<div class="container">  
  <div class="dateContainer">  
    <p class="date"> 2020-12-04 </p>  
  </div>  
</div>
```

Even though class **date** is nested inside **dateContainer** it will still get the color **blue**. Why? Because the syntax **.container .date { }** is applied to all children, no matter how far down the **DOM** they are nested... and element with class **date** is a children of container (but not a direct one)

- **With ">" sign. This will be applied to children who are directly related to the parent.**

To better illustrate this point another type of selector combination needs to be shown.

The first one will be applied to all class **date** which are inside class **container**. No matter how many layers below... But what if we want to apply it only to direct children? What does that mean? It means that only elements that are directly nested inside will get that style. This needs an example.

```
.container > .date {  
  color: blue;  
}
```

If we wrote this selector instead of the one with only space. The style color: blue would only be applied to this HTML:

```
<div class="container">  
  <p class="date"> 2020-12-04 </p>  
</div>
```

But not this one:

```
<div class="container">  
  <div class="dateContainer">  
    <p class="date"> 2020-12-04 </p>  
  </div>  
</div>
```

Meaning that combining selectors with **>** is more secure as it is only applied to **directly nested** elements

Remember 😊

This might be difficult to understand at first but just remember a few things:

- Selectors can be applied to tags (div, p, etc..)
- Selectors can be applied to classes

- Selectors can be applied to ID's
 - Selectors can be applied to **pseudoclasses**
 - **Pseudoclasses** - special state of HTML elements. (hover, focus etc)
 - Selectors can be combined!
 - Applying the styles to **ALL** children (with space)
 - Applying styles to **direct children only** (with > sign)
-

CSS Properties

There are many types of CSS properties all of which can be found on the web. **You don't need to memorize them.** But once you start using them often they will memorize automatically. Here are some most popular types:


Text Formatting

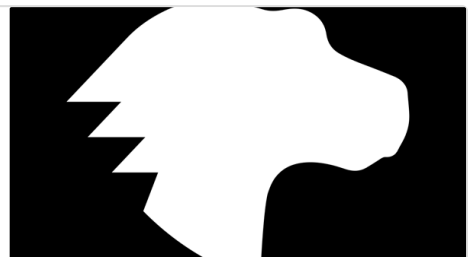
- Changing color
- Font
- Font size
- Font weight or style

For this I'd recommend to use this place:

Styling text

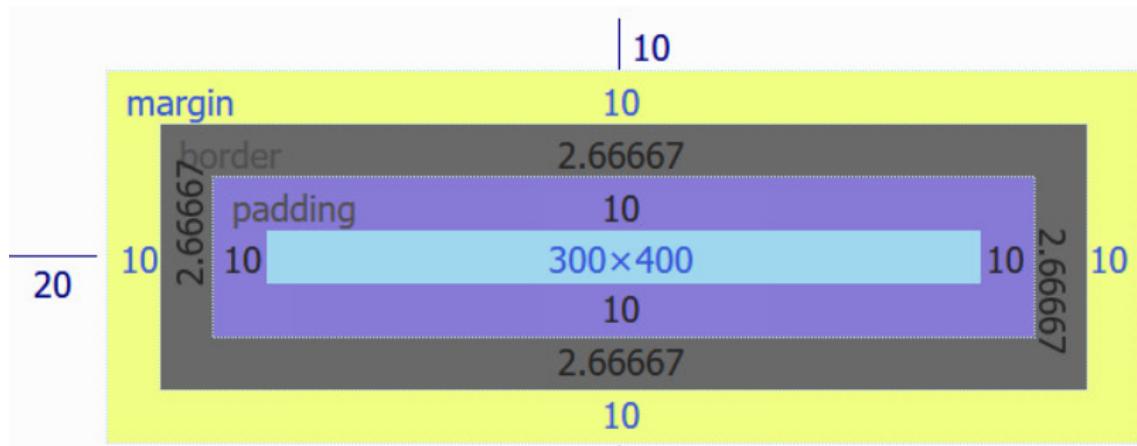
With the basics of the CSS language covered, the next CSS topic for you to concentrate on is styling text - one of the most common things you'll do with CSS. Here we look at text

 https://developer.mozilla.org/en-US/docs/Learn/CSS/Styling_text



It has guides how to style links, lists and text. Also includes a tutorial about fonts if you ever need that.

CSS Layout



For reference this visual representation (which can be seen in the elements tab in the browser using F12) is very helpful. You should remember, know these properties:

- Padding (distance between content and the border)
- Border
- Margin (distance between border and other content.
- Width and height

Using these properties we can style individual blocks or tags in an HTML document. But we also need to style collections of blocks.

Positioning

We use this CSS property to determine the position of the element:

- **Static** (default) → puts the element according to its normal layout flow
- **Relative** → allows you to move the element relative to its position in normal flow (for example using it to overlap with other elements)
- **Absolute** → moves the element completely out of the normal layout flow
- **Fixed** → fixes the element according to the viewport not another element (let's stay it's stay fixed relative to your website "view")
- **Sticky** → newer property that acts as **position: static** until it is hidden at which point it acts like **position: fixed** (might be used for sticky headers that stay in their place even if you scroll)

Float

This property allows an element to change its normal flow behavior. For example with it you can move elements to the left, right and remove them from normal (default) flow and "float" above other items. It has four possible values:

- left
- right
- none
- inherit

Overflow

Sometimes some elements have too much content → we can say that the content is **overflowing** from its defined size. We can use this property to determine what to do if the content does not fit in:

- overflow: hidden → hides the excessive or overflowing content
- overflow: visible → allows the content to overflow and be visible outside the box
- overflow: scroll → the elements is not visible outside a box but gets a scroll bar which a user can scroll and see the rest.

Flexbox and Grid

Although you can use **position**, **display**, **float** properties to create layouts a few are really helpful and make it easy for us to lay things out in one dimension (either as row or column).

Flexbox

To use flexbox you must apply **display: flex** on a parent element (for example a div) and then its children will become flex items. By default flex lays out elements in a horizontal order. In order to change that we can add the **flex-direction: column** property (this will layout elements in one column, so vertically) and **flex-direction: row** (the default one that will layout in one row or horizontally).

Grid

We can think of flexbox that is designed for one-dimensional layout and grid as **two-dimensional** → with **rows** and **columns**.

The same as with flex we need to add **display: grid;** on a parent element. we also need to define rows and columns with grid, for example:

```
.container {  
  display: grid;  
  grid-template-columns: 1fr 1fr 1fr;  
  grid-template-rows: 100px 100px;  
}
```

This code will create a container with 3 equal columns (1fr 1fr 1fr) and with 2 rows each with a 100px height.

Honorable mentions

Colors in CSS

We can write colors in CSS using a few ways:

- words (red, blue)
- hexadecimal number (starting with # for example #2233AA)
- as a **rgb** color (rgb(22,33,99))

For all this use color pickers online (any you find are fine)

CSS Units

We can measure things in a few ways in CSS (it is best to use relative measurements which are not constant or static)

- px → pixels
- % → percent of elements free space
- vh → view height (percent of current window height → 1vh is 1 percent)
- vw → view width
- mm, cm → metrical units
- em, rem → em (units relevant to parent font-size) and rem (unit relative to font-size of root)

Order of rules

Rules in CSS have an order that they are applied in (meaning that some are superior to others). Starting from the most important rules first the order is:

- rules with **!important** keyword
- rules defined in style attribute (in html)
- rules with id selector **#id1**
- rules with class selector **.c1**
- rules with element selector **div**

Inheritance

It's also important to know that elements inherit style from each other. It does not apply to all styles but to many. If a style is set on a parent in most cases the child element will inherit that style

Responsive web design

If you want to write responsive web pages you need these things

1. Add a tag in the <head> section of your HTML

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

2. Use relative sizes (em, rem for text and vw, wh or % for containers like divs). If you use pixels a size of 1000px can be too high for smartphones to be visible.

3. If possible avoids sizes at all, write CSS which aligns your content automatically (for example using flex and grid → you wont need width and height properties in your elements)

4. To make it responsive use media queries

```
/* Screens which are lower than 1000px width */
@media screen and (max-width: 1000px) {
  display: hidden;
}
```

Personal recommendations


- Don't overuse !important parameter
 - Write styles in a separate file
 - Try to write as minimal amount of CSS and as readable as possible. CSS code can get bulky really fast and keeping the structure and styles minimal can help you save time later. So structure it wisely.
 - Write mobile first websites (if it works on the mobile it will be easier to work on a desktop)
 - Test using F12
 - Google, search, investigate and don't be scared to experiment 😊
-

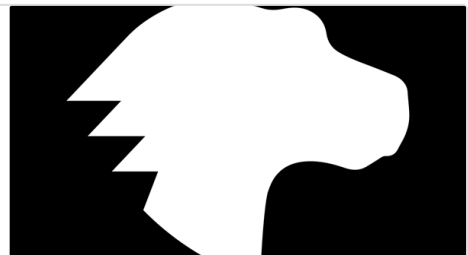
Material for further reading

Some material about this (it has great examples which cannot be shown in a PDF):

Introduction to CSS layout

This article will recap some of the CSS layout features we've already touched upon in previous modules - such as different values - and introduce some of the concepts we'll be


 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Introduction



CSS layout basics

CSS layout

At this point we've already looked at CSS fundamentals, how to style text, and how to style and manipulate the boxes that your content sits inside. Now it's time to look at how to place

 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout

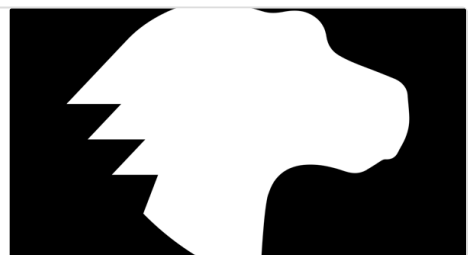


More guides about CSS layout

Flexbox

That concludes our tour of the basics of flexbox. We hope you had fun, and will have a good play around with it as you travel forward with your learning. Next, we'll have a look at


 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Flexbox

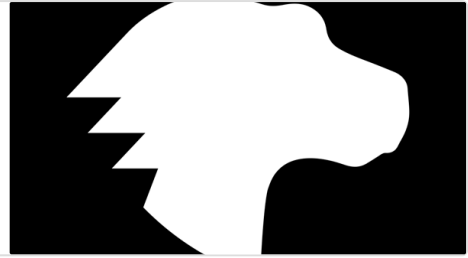


Flexbox guides and references

Grids

In this overview we have toured the main features of CSS Grid Layout. You should be able to start using it in your designs. To dig further into the specification, read our guides

 https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Grids



Grid references, overview, guides and tutorials