<https://developer.mozilla.org/en-US/docs/Web/HTML/Element>

<h1>Macbi' Challenge for Friday, January 20th</h1>

<h1 style=”font-family: sans-serif”>Macbi' Challenge for Friday, January 20th</h1>

--- style is an attribute. style=”” no white space between. ---

Add css style directly to html element

Can add more than 1 property to the element. Use ; between each property

COLOR

Hex code : #ABCDEF AB=red CD=green EF=blue

--- google color picker --

Graphical user interface

Description automatically generated with low confidence

Format Document

Shift + option + F



Extension – Prettier Don’t forget to set the format default.

<a> element = anchor element

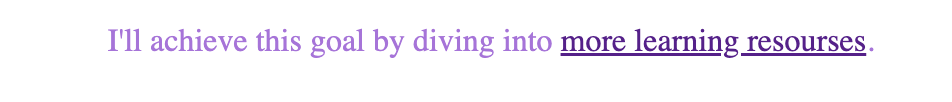
Add link

<p *style*="font-family: ;sans-serif; text-align:center; color:#a771da">

I'll achieve this goal by diving into

<a *href*="https://www.google.com">more learning resourses</a>.

</p>



When not using inline styles (via the style attribute), CSS code typically looks something like this:

1. p {
2. font-family: sans-serif;
3. text-align: center;
4. }

This code is formatted to be more readable. Theoretically, you could also write it like this:

1. p {font-family: sans-serif;text-align: center;}

But of course such kind of code is way harder to understand and maintain, hence we typically go for the more readable version.

Here are a couple of conventions about CSS code formatting, which you should keep in mind:

* The selector (p in the above example) and the opening curly brace typically go into the same line
* You then have one CSS property + its value per line
* Every line MUST end with a semi-colon
* The lines are indented (automatically, via the "Format Document" shortcut or because you pressed the TAB key on your keyboard)
* The closing curly brace goes into a separate line, with no indentation

<style>

h1 {

font-family: sans-serif;

text-align: center;

color: red;

}

p {

font-family: sans-serif;

text-align: center;

color: #a771da;

}

</style>

<h1>Macbi' Challenge for Friday, January 20th</h1>

<p>

Learn about the basics of web development - specifically dive into HTML & CSS.

</p>

<p>

I'll achieve this goal by diving into

<a *href*="https://www.google.com">more learning resourses</a>.

</p>

HTML Document skeleton

<!DOCTYPE *html*>

<html>

<head>

Put <style>, <title>

</head>

<body>

Put <h1>, <p>, …

</body>

</html>

<title> 🡪 title on the tap

Useful short cut

* + Can drag and drop the marked phrase
  + Use option key + arrow keys to move the marked phrase

"**Comments**" in code.

As a developer, you can add extra, human-readable comments into your HTML or CSS code which will be ignored by the browser but can help you or other developers understand your code.

Here's how you would add a comment in HTML:

1. <body>
2. <h1>This is a main title!</h1>
3. <!-- This is a comment - the browser ignores it. It won't show up on the user's screen -->
4. </body>

Comments are added with help of the special <!-- opening and --> closing tags. They are only visible in your source code, not on the rendered page.

You can also add comments in your CSS code:

1. p {
2. font-family: sans-serif; /\* Switch to sans-serif instead of serif \*/
3. }

In CSS, you create comments via the /\* \*/. Again, you can add extra annotations for other developers (or yourself) with help of comments - the browser will ignore them, they hence won't affect your page styles.

Comments are also not just used for adding extra information but also for "**commenting out**" unused code.

In Visual Studio Code, there also are **shortcuts** for quickly adding or removing comments around the code in a selected line - simply search for the "Toggle Line Comment" shortcut

**Psudo selector**

a:hover {

text-decoration: underline;

}

a:hover

Separate CSS to another file

Link CSS file with HTML file

<head>

<link *href*="styles.css" *rel*="stylesheet">

<title>My daily challenges</title>

</head>

<link *href*="styles.css" *rel*="stylesheet" />

< ……../> is OK too.

Link to css file have to be below the link for fonts!!!!

**Set id for HTML element**

- id have to be unique

- start with # in css file

#todays-challenge{

color: **rgba**(16, 16, 196, 0.721);

font-weight: bold;

font-size: 45px;

}

**Google fonts**

[**https://fonts.google.com/about**](https://fonts.google.com/about)

* select the fonts you like
* copy the link and paste in HTML file
* copy the CSS rules and paste in CSS file

<head>

<link *rel*="preconnect" *href*="https://fonts.googleapis.com" />

<link *rel*="preconnect" *href*="https://fonts.gstatic.com" *crossorigin* />

<link

*href*="https://fonts.googleapis.com/css2?family=Raleway:ital,wght@1,100&display=swap"

*rel*="stylesheet"

/>

**Understanding How HTML & CSS Handle Text & Whitespace [Day 5]**

In this lecture, we'll explore two main concepts.

* How browsers handle "whitespace" (line breaks and indentation)
* How you can output special characters (e.g. "<") as text in HTML documents

How Browsers Handle Whitespace

In both HTML and CSS (and later also in "JavaScript"), as a developer, you typically try to format and structure code such that it is readable (for humans).

For example, the following two snippets contain the same code and hence would lead to the same result. The browser would understand both, but they are not equally readable / understandable for us humans:

**1) No formatting**

1. <html><head><title>A test </title><style>h1{color:red}</style></head><body><h1>Hi there!</h1><p>This is some text...</p></body></html>

**2) Formatting with line breaks and indentation (i.e. lots of "whitespace")**

1. <html>
2. <head>
3. <title>A test </title>
4. <style>
5. h1 {
6. color: red;
7. }
8. </style>
9. </head>
10. <body>
11. <h1>Hi there!</h1>
12. <p>This is some text...</p>
13. </body>
14. </html>

By default, the browser (typically - there are few exceptions, which we'll explore later) **ignores line breaks and indentation** in your HTML and CSS code. That's why, as a visitor of the site, you will see the same result for both snippets.

Since the result is the same, but we as a developer are a human, we typically go for the second approach - using lots of indentation and line breaks to structure and organize our code.

How To Output Special Characters In HTML

When writing HTML code, characters like *"<"* and *">"* obviously have a special meaning: They mark the beginning and ending of HTML tags.

But what if you would want to output the *"<"* and *">"* characters or a complete HTML tag as text on your website? Like on this site here (yes, the site on which you currently are). You can read the code snippets above just fine - because they are output as plain text (they are NOT interpreted as HTML by the browser that loaded this page).

There are two main ways of achieving this:

* You can use the special <pre>...</pre> tags (for "preformatted text") - these tags wrap any text (that may include HTML code) and "tell the browser" to **output it as plain text** (i.e. NOT interpret it as HTML code). When using <pre>, whitespace is also preserved and NOT ignored (as it normally would be)
* Alternatively, if you simply want to output the "<" character (e.g. in some math formula that should be shown on your page), you can use some special "shortcuts" (so-called "HTML entities") in your HTML code:
  + E.g. if you write &gt; in your HTML code, the browser will output the *">"* (**g**reater **t**han) symbol
  + &lt; => *"<"* (**l**ower **t**han)

**Add images**

Add <img> attribute in body part

<img *src*="challenges-trophy.jpeg" *alt*="A trophy">

alt is for screen reader

img {

width: 200px;

height: 200px;

border-radius: 100px;

}

Add spacing around the content

body {

text-align: center;

background-color: **rgba**(232, 210, 223, 0.781);

margin: 50px;

}

Add margin 50 px at the left, right, top and bottom

**A Word About File Name Conventions [Day 5]**

In (web) development, we have a lot of **rules and conventions** when it comes to naming things.

For example, we named our main HTML file index.html. This is NOT something you have to do - it's just a common convention. And some hosting providers might require that name, in order to serve your files successfully. But in general, you could've named it My First File.html as well. Nonetheless, it is recommended to stick to such common conventions.

The question is: **How do you name your second, third etc. HTML files? And how you should you name your CSS files?**

For **HTML files**, it's generally a good idea to give them names that **describe the main purpose** or content of the page that will be loaded.

For example, if you have a HTML file that will display the shopping cart content of a user, cart.html might be a fitting name. The HTML file that is responsible for displaying a bunch of online shop products might be named products.html.

For **CSS files**, you typically either have a file that belong to a specific HTML file or you have global CSS files (that are used in multiple / all HTML files):

* For page-specific CSS files, it's a good idea to repeat the HTML filename (e.g. cart.css holds the styles for cart.html).
* For CSS files that belong to multiple HTML files, you might want to choose general names like base.css or describe the general purpose of the HTML files to which the CSS file belongs, like online-shop.css for both the cart.html and products.html files.

There's also one **important characteristic**which you maybe noticed about all these filenames: They are **all lowercase.**

And that's important! Whilst it's technically not required, it is a **very common convention** that you name your files **all-lowercase**, with **no special characters** except for dashes (-). If your file name consists of multiple words, you should NOT separate them with blanks (whitespace) but instead use dashes. So use online-shop.html instead of Online Shop.html.

**Live server extension** : not need to refresh the page after modification (auto-refresh)

* + Right click on html file and select open with live server

**The Development Server & The Local Website Address [Day 6]**

In the previous lecture, we started a **local development web server** via the "Live Server" Extension for VS Code.

What Is A "Development Web Server"?

It's a "local development web server" because it's a web server software that serves the website locally, on our machine. It's **NOT exposing**our machine or the website to the internet - you can only visit and reach it locally (i.e. from your machine).

This "web server software" (i.e. the "Live Server" extension in this case) is a software that does actually listen for incoming HTTP requests and send back appropriate responses (that contain the HTML code for example). Remember that **request + response**image form the first course section!

*Later in the course, we'll also set up our own web server that is able to do more things than just send back pre-defined HTML code. We'll dive into the creation of our own backend and our own web server with a technology called "NodeJS" from section 16 on.*

What's This Address: 127.0.0.1?

As mentioned above, this development web server **hosts** (= provides / serves) the website **from our local machine to our local machine**.

In section 1, you learned that users enter addresses (URLs) into the browser address bar to reach a website and send such a request. You also learned that the human-readable "domains" (like academind.com) are translated to IP addresses which act as unique identifiers of machines connected to the internet.

**127.0.0.1 is such an IP - though it's a special one!**

127.0.0.1 is a special IP, that's **reserved to your local machine**. And it's the **local machine for everyone**!

If I type 127.0.0.1 into the browser, I connect to **my** local machine (if it's running a local web server). You reach **your** machine.

It's an IP address that's NOT assigned to other machines in the world wide web - instead it's reserved as a "placeholder" that always points at your local machine. It exists for use-cases as we have it here: For development on our local machine, where we want to test our website with help of a local development server. I.e. we can test it locally without exposing it to the entire world yet.

There also is an **alias** (basically like a "special domain name") that you can use locally, instead of 127.0.0.1: localhost. You can also enter localhost into your browser and it will be the same as if you entered 127.0.0.1. So localhost:5500 is a replacement for 127.0.0.1.

What's This Thing: :5500?

The :5500 part is a so-called "**port**".

Ports are another concept from the networking world. The idea is, that a machine can **expose different processes** (e.g. different web servers serving different websites) via different ports.

So a single machine could host / provide three different websites on three different ports. The IP address of the local machine would always be the same (127.0.0.1) but every website would have its own port (e.g. 5500, 3000, 8080).

If you move a website to some machine that IS exposed to the world wide web (i.e. you publish it, you don't run it via a development web server on your local machine anymore), then the website is reached via the IP address of that remote machine. Or, typically, via a domain that's pointing at that IP address.

In addition, this port concept also still exists: When exposed to the world wide web, websites are typically served on ports 80 (HTTP) or 443 (HTTPS). You don't need to worry about this right now though. When publishing a website (covered later in the course), the different hosting providers typically take care about exposing the right ports automatically.

When working on your local machine, you don't use these "common ports" (80, 443) since you're not exposing the website to the world wide web anyways. Instead, you can use **ANY ports that are typically not used by any other processes** - 5500, 3000 or 8080 are common choices because they aren't typically used by other processes.

That's why the "Live Server" extension does use port 5500 for serving your website locally.

And you target a specific port by adding :<port-number> after the domain or IP address. That's why 127.0.0.1:5500 resolves to that locally served website. Alternatively, since 127.0.0.1 is aliased with localhost, you could also enter localhost:5500.

You can try this with other websites, too!

Try visiting academind.com:443 or academind.com:80 => You will see the regular Academind website. Of course you don't need to add the extra port information in the URL though - since 80 and 443 are the defaults, the browser will use these ports automatically, if you enter a website address.

**List**

* Order list <ol> </ol>

<ol>

<li>Monday</li>

<li>Tuesday</li>

<li>Wednesday</li>

</ol>

* Unorder list <ul> </ul>

<ul>

<li>Apples</li>

<li>Bananas</li>

<li>Tomatoes</li>

</ul>

Inheritance : (selected) container rules apply to descendants

Cascading style sheets : Multiple rules can be applied to the same element. The latter rule win.

Specificity : More specific selector’s wins over less specific selector

**The CSS box model**

Padding : space between the content and the box

You can define the padding of 4 sides from top , right, bottom and left.

Border :

Margin : space between the two nearby elements

You can define the margin of 2 sets, top-bottom and left-right.

p {

padding: 2px 5px 2px 5 px;

border: 2px dashed black;

margin: 8px 6px;

}

**Adding structure to a webpage**

<https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure>

**Graphical user interface, text, application, email

Description automatically generated**

<header> </header> such as h1, a, img

<main> </main> such as p, ol, ul

**Selectors & Combinators**

Selectors

Type :elementname

ID : #id have to be unique, can use with only one element.

Group : elementname

Class : .class class selector can be used with many elements.

Combinators

Descendant : li p : All p with li as ancestor

Child : h2 > p : Only p which are direct children of h2

**Block vs Inline elements**

Common inline elements

<a> : Link to other website/page of current website

<button> : Clickable button to submit forms

<img> : Embed image into HTML page

<span> : Non-semanic inline container to mark-up text for specific style

Etc…

<https://developer.mozilla.org/en-US/docs/Web/HTML/Inline_elements>

2 types of inline elements

Replaced : vertical margin can be applied <img>

Non-replaced : vertical margin cannot be applied <a>

We also can use negative margin too !

Block elements

<p>

display property

-define the property : inline, block, inline-block …etc.

Inline element : cannot define top-bottom margin/padding

display: inline-block

Margin collapsing : only happen with a block element (vertical margin)

The bigger margin wins.

Box shadow property

box-shadow: 5px 5px 5px grey;

make a box shadow; x-axis size, y-axis size, blur size, color

<div> : non-semantic block element

<span> : non-semantic inline element

**A List Of All CSS Selectors [Day 8]**

Thus far, we already learned about a lot of key CSS selectors. Here's a list / summary of all the CSS selectors that you can use in CSS (including some, which we haven't seen yet):

* **Tag type selector**
  + CSS: p { ... }
  + Would select this HTML element for example: <p>Some text...</p>
  + This selector selects all HTML elements that are of this tag type
* **ID selector**
  + CSS: #some-id { ... }
  + Would select this HTML element for example: <h1 id="some-id">...</h1>
  + This selectors selects the element that has this ID on it (should only be once per page)
* **Class selector**
  + CSS: .some-class { ... }
  + Would select this HTML element for example: <h1 class="some-class">...</h1>
  + This selector selects all HTML elements that have this class on them
* **Attribute selector***(new)*
  + CSS: [src] { ... }
  + Would select this HTML element for example: <img src="...">
  + This selector selects all elements that have this HTML attribute on them
* **Universal selector** *(new)*
  + CSS: \* { ... }
  + Would select this HTML element for example: <p>....</p><img ...>
  + This selector selects ALL HTML elements (directly, not through inheritance but as if you would target them all individually)
* **Grouping selector / selector list**
  + CSS: p, .some-class { ... }
  + Would select this HTML element for example: <p>...</p><h2 class="some-class">...</h2>
  + This selector selects all elements that match the individual selectors in that list
* **Combined selector**
  + CSS: p.some-class { ... }
  + Would select this HTML element for example: <p class="some-class">...</p>
  + This selector selects all elements that meet both conditions (i.e. "<p>" elements with "some-class" class on it, in this example)
* **Pseudo selector**
  + CSS: a:hover { ... }
  + Would select this HTML element for example: <a>...</a>*(when the user hovers over it)*
  + This selector selects all elements that meet this "pseudo state" (i.e. all links that are hovered in this example)

\*\*\* <a href=” “ target=”\_blank”> </a> download in a new tab

<a href=” “ target=”\_self”> </a> download in a same tab

<strong> </strong> the reader is using strong voice (emphasize)

Favicon : to make an icon in front of a title

Flex box : set on the parent element such as <header>. Use the id for easier styling

Display: flex;

Flex-direction: ;

Flex-wrap: ; = wrap when adjust screen size

Justify-content: ;

Adding background image

#hero {

height: 800px;

background-image: **url**("/images/places/ocean.jpeg");

background-position: center;

background-size: cover;

}

Positioning elements

#hero-content {

width: 900px;

background-color: **rgba**(51, 47, 47, 0.8);

box-shadow: 2px 4px 8px **rgb**(68,67, 67);

border-radius: 8px;

text-align: center;

padding: 50px 0;

margin: 0 auto;

**position: relative;**

**top: 200px;**

}

Position element

absolute : is positioned relative to the nearest positioned ancestor (instead of positioned relative to the viewport, like fixed). However; if an absolute positioned element has no positioned ancestors, it uses the document body, and moves along with page scrolling.

relative : Setting the top, right, bottom, and left properties of a relatively-positioned element will cause it to be adjusted away from its normal position. Other content will not be adjusted to fit into any gap left by the element.

Static : default

fixed : is positioned relative to the viewport, which means it always stays in the same place even if the page is scrolled. The top, right, bottom, and left properties are used to position the element.

Sticky : is positioned based on the user's scroll position. A sticky element toggles between relative and fixed, depending on the scroll position. It is positioned relative until a given offset position is met in the viewport - then it "sticks" in place (like position:fixed).

<https://codingbff.medium.com/difference-between-css-position-absolute-versus-relative-35f064384c6>

header {

display: flex;

justify-content: space-between;

align-items: center;

padding: 15px 60px;

position: absolute;

**width: 100%;**

**box-sizing: border-box;**

**}**

Red- for set the property expand 100% of html width. Then, box-sizing to expand only inside border, not over the border.

object-fit: fill;

to make picture/object fill in all space and not distort.

object-fit: contain;

will leave a space

background: **linear-gradient**(45deg, **rgb**(227, 255, 253), **rgb**(31, 130, 123))

add color gradient in background color deg = degree

overflow: hidden;

sets the desired behavior for an element's overflow — i.e. when an element's content is too big to fit in its [block formatting context](https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Block_formatting_context) — in both directions.

https://developer.mozilla.org/en-US/docs/Web/CSS/overflow

object-fit: cover;

 sets how the content of a [replaced element](https://developer.mozilla.org/en-US/docs/Web/CSS/Replaced_element), such as an [<img>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img) or [<video>](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/video), should be resized to fit its container.

https://developer.mozilla.org/en-US/docs/Web/CSS/object-fit

display: flex;

flex-direction: column;

justify-content: space-between;

CSS Grid

display: grid;

grid-template-columns: 1fr 1fr;

gap: 100px 200px;

fr = fraction (each colum has 1 fraction), can also use px

gap: distant between row , distant between column

li:nth-of-type(3) {

background-color: aquamarine;

grid-column: 1 / span 2; /\* or grid-column: 1 / 3; \*/

}

Select the number of content

Grid-column: how many column / span across 2 columns

Working with Unicode UTF-8

<https://www.w3schools.com/charsets/ref_utf_arrows.asp>

HEX 2190 html &#x2190 🡸

**Flexbox & the Grid [Day 20]**

#### Beyond the Basics

In this module we explored the position property and understood that Flexbox and the Grid are great technologies to easily change the layout of elements.

With the concepts covered in this module, you are ready to go to use these technologies in your own projects

In case you want to dive deeper here, below you find dedicated resources which go beyond the core focus of this course, feel free to explore these.

---

More about the position property => <https://academind.com/tutorials/the-position-property/>

Flexbox - Flex Container => <https://academind.com/tutorials/flexbox-basics-container/>

Flexbox - Flex Items => <https://academind.com/tutorials/flexbox-flex-items/>

Comparing Flexbox and the Grid => <https://academind.com/tutorials/css-grid-vs-flexbox/>

<article> </article> element

Responsive design

Comparing Units

px : easy to understand and translatable. BUT limit user focus & not scalable

% : relative to parent element size. BUT hard to manage due to cascading nature

em : size is relative to font-size. BUT hard to manage due to cascading nature

rem : size is relative to root element (HTML element)’s font size. Preferred choice if applicable

16px ~ 1 rem 16px is usually a browser setting. if you set 2 rem then = 32 px

Media queries

\*\* put it at the bottom for apply the rule over another above rule\*\*

@media (max-width: 48rem) {

#main-header nav {

display: none;

}

main h1 {

font-size: 1.5rem;

}

#latest-products ul {

grid-template-columns: 100%;

}

}

common breakpoints for media queries

Portrait

smartphone 480px

tablet 768px

Landscape

notebook 1024px

desktop computer 1200px

TV >1200px

max-width : start from bigger to smaller screen size

min-width : start fromm small creen to bigger screen size

Side drawer & Hamburger icon

1. Create clickable hamburger button in mobile view

HTML

<nav>…

</nav>

<a *href*="#side-drawer" *class*="menu-btn"> 🡪 link to #side-drawer that created below

<span></span>

<span></span>

<span></span>

</a>

CSS

.menu-btn {

width: 3rem;

height: 3rem;

display: flex;

flex-direction: column;

justify-content: space-around;

}

.menu-btn span {

width: 100%;

height: 3px;

background-color: white;

}

@media (max-width: 48rem) {

.menu-btn {

display: flex;

}

.

.

.

1. Open side drawer on the first button click
2. Close side drawer on second button click

HTML

<aside *id*="side-drawer">

<header>

<a *href*="#" *class*="menu-btn"> *<!-- # is the main page -->*

<span></span>

<span></span>

<span></span>

</a>

</header>

<nav>

<ul>

<li>

<a *href*="">Browse Meals</a>

</li>

<li>

<a *href*="">My Orders</a>

</li>

</ul>

</nav>

</aside>

\*\* copy nav inside #side-drawer

CSS

#side-drawer {

width: 100%;

height: 100%;

background-color: **rgb**(29, 26, 27);

position: fixed;

top: 0;

left: 0;

display: none;

}

#side-drawer:target {

display: block;

}

#side-drawer header {

height: 5rem;

display: flex;

justify-content: flex-end; 🡪 move hamburger to the right side

align-items: center;

padding: 0 5%;

}

#side-drawer ul {

list-style: none;

margin: 0;

padding: 4rem 1rem;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

}

#side-drawer li {

margin: 1rem 0;

}

#side-drawer a {

color: **rgb**(253, 239, 213);

font: 2rem;

}

**Introducing the "z-index" Property [Day 24]**

#### The z-index - What & Why?

We learned about the document flow and how to change the default positioning of HTML elements with the position property.

By changing the default element position, you might come across a situation were one element is overlapping the other one and where you want to control, which element is positioned above the other.

For such cases, the z-index is required.

**How does it work**

The z-index , a CSS property, defines the z-order of a positioned element (i.e. an element with the position property applied with a value different than static). The z-order refers to the z-axis, so it controls how elements are stacked above it each other if applicable.

The default value for HTML elements is set to auto, which is aquivalent to 0. Positioned elements with a higher z-index are positioned above elements with a lower value for the z-index.

**Code example**

HTML

1. <body>
2. <div id="first">Element 1</div>
3. <div id="second">Element 2</div>
4. </body>

CSS

1. div {
2. width: 200px;
3. height: 200px;
4. text-align: center;
5. color: white;
6. }
8. #first {
9. background-color: rgb(55, 117, 209);
10. }
12. #second {
13. background-color: rgb(233, 137, 59);
14. }

By default, both elements follow the document flow, element 1 is displayed first, element 2 is displayed second, one after another.

Now add the following code to #second:

1. position: absolute;
2. top: 0;
3. left: 0;

Element 2 is taken out of the document flow and displayed on top of element 1.

Adding z-index: 1 to #first does not change the order of the elements along the z-axis, as the z-index only has an impact on positioned elements, excluding the default value position: static.

Adding position: relative to #first, will display element 1 above element 2 again.