**Lab 2 - CSS Basics**

The goal of this lab is to help you familiarize with CSS. We will be using minimal HTML boilerplate and focus mainly on CSS. Each section of these instructions corresponds to one <article> within the provided HTML. Your goal is to add and edit **index.css** to style the provided HTML content in **index.html**.

***TIP:*** *To comment out the line of code in VS code you can use* ***ctrl + /*** *(or* ***cmd + /*** *on MacOS)*

***TIP:*** *You can repeat the same CSS rule multiple times and only the last defined one will apply, e.g. In this case the final margin will be set to 30px:*

.class {

margin: 20px;

margin: 10px;

margin: 30px;

}

**I. Selectors and combinators**

<https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors>

1. Selectors
   1. Using the **type** selector, add **margin**, **padding** and **border** to each <article> on the page, e.g.

margin: 20px 0;

padding: 20px;

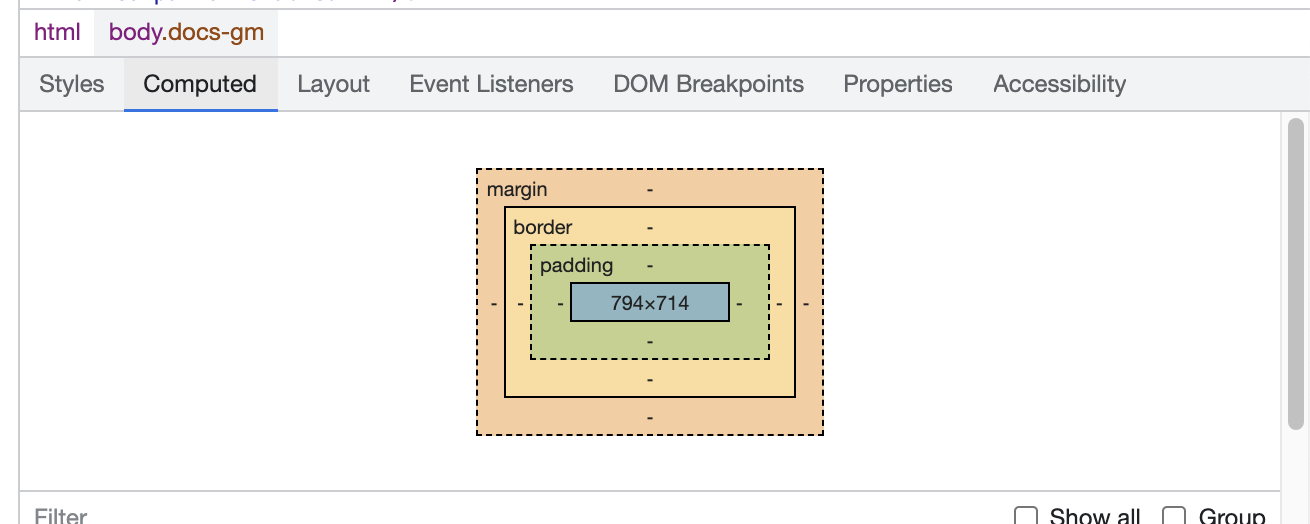
border: 1px solid black;

* 1. Using a **class** selector, change the **font color** of the element with class style-me1 to **red**
  2. Using an **id** selector, change the **font color** of the element with id style-me2 to **blue**
  3. Using a **specific** selector, change the **font color** of the element that has both a class of style-me1 and an id of style-me2 to **purple**

***TIP:*** *Note that in normal situations having duplicate ids is a mistake - there should not be duplicate ids on the same page. We are doing it here only for the simplicity of the exercise.*

1. Combinators
   1. Using a **descendant combinator**, style all the <p> elements that are descendants of .by setting their **background** to **red**.
   2. Using a **child combinator,** style all the <p> elements that are direct children of .my-children-are-styled, by setting their **background** to **lightgreen**.

**II. Box model and common properties**

In this section you will be editing CSS rules for the .box class. Use an inspection tool throughout this exercise - take a look at both **Styles** and **Computed** tabs.

1. Set **margin** and **padding** on an element with class .box :
   1. First set a **background colour** for your box to easily see where it starts and finishes
   2. Now use different margin and padding syntaxes - **4-value** syntax, **2-value** syntax and **1-value** syntax. Use **pixels** as units for margin and padding but do not hesitate to try out other types of units including em and %.
2. Practice setting **width** and **height** :
   1. Set width and height using different dimension units: px, rem, vh, vw.   
      *vh stands for viewport height, and vw stands for viewport width.*
   2. Observe how the third box retains height styled via the **style** attribute. Why is this? Force your CSS rules to take precedence using the !important operator.
   3. Try resizing the browser window while the box size depends on the vw and vh units. What can you observe?
3. Practice adding a **border** :
   1. Set a **border** on the .box element: try 2 or 3 different border **styles** and **colors**
   2. Make the border 10px thick and solid, and observe how thickness of the border moves surrounding elements in each direction.
4. The **box-sizing** property :
   1. Test the two different values of box-sizing. How do they affect the width and height of the box elements?
5. Use **overflow** to define the behaviour when content does not fit inside its container :
   1. Make text overflow **outside** of the element by setting width and height to smaller values, so the text in the second box goes over the box borders.
   2. Set the **overflow** CSS rule for the .box class - test several different values. How do they affect the box?
6. Add an **outline** :
   1. Set a wide outline (10px) on the .box. Observe that setting the outline does not shift the other elements around, like with borders.
7. Use **border-radius** to affect the box corners :
   1. Set border-radius on the .box element. Start by trying out different values expressed in pixels, e.g. 20px or ems e.g. 1em
   2. Setting border-radius to 50% together with width=height will make the element look like a circle. If the width is different from height, then the element will become an oval. *Note that this is just a visual effect of the border. When you use the Inspector tool, the element is still a rectangle.*
   3. Increase the padding so the text inside the box is not cut off by the border corners
8. Adding a **background image** :
   1. Set a background image for the box using the background property. Image to use is located in ./assets/images/img1.png file
   2. Set background-size to `cover`. Also experiment with other values.
   3. Set background-repeat to `no-repeat`. Also experiment with other values.
   4. Set background-position to `center`. Also experiment with other values.

**III. Positioning**

For this section, you will be editing the classes for the third article in the given HTML to apply different positioning rules.

1. **Absolute** positioning :
   1. Use CSS to apply absolute position on the .position-absolute class
   2. On the same class, add a **top** position of 0 to fix this element to the top of the page (also try changing **top** to **bottom**)
   3. Add a **background color** to make finding the absolutely positioned elements easier
2. **Relative** positioning for absolute elements :
   1. Set **relative** position on the .position-relative class
   2. Add a different **background colour** to the relatively positioned container
   3. How is absolute positioning **within** relative different from absolute that is **outside** of relative?
   4. Add a left position value to the .position-absolute class and observe the difference. Add a right position value as well – what does this do?

***TIP:*** *You can use ctrl+f (cmd+f on MacOs) to search for specific content on the page.*

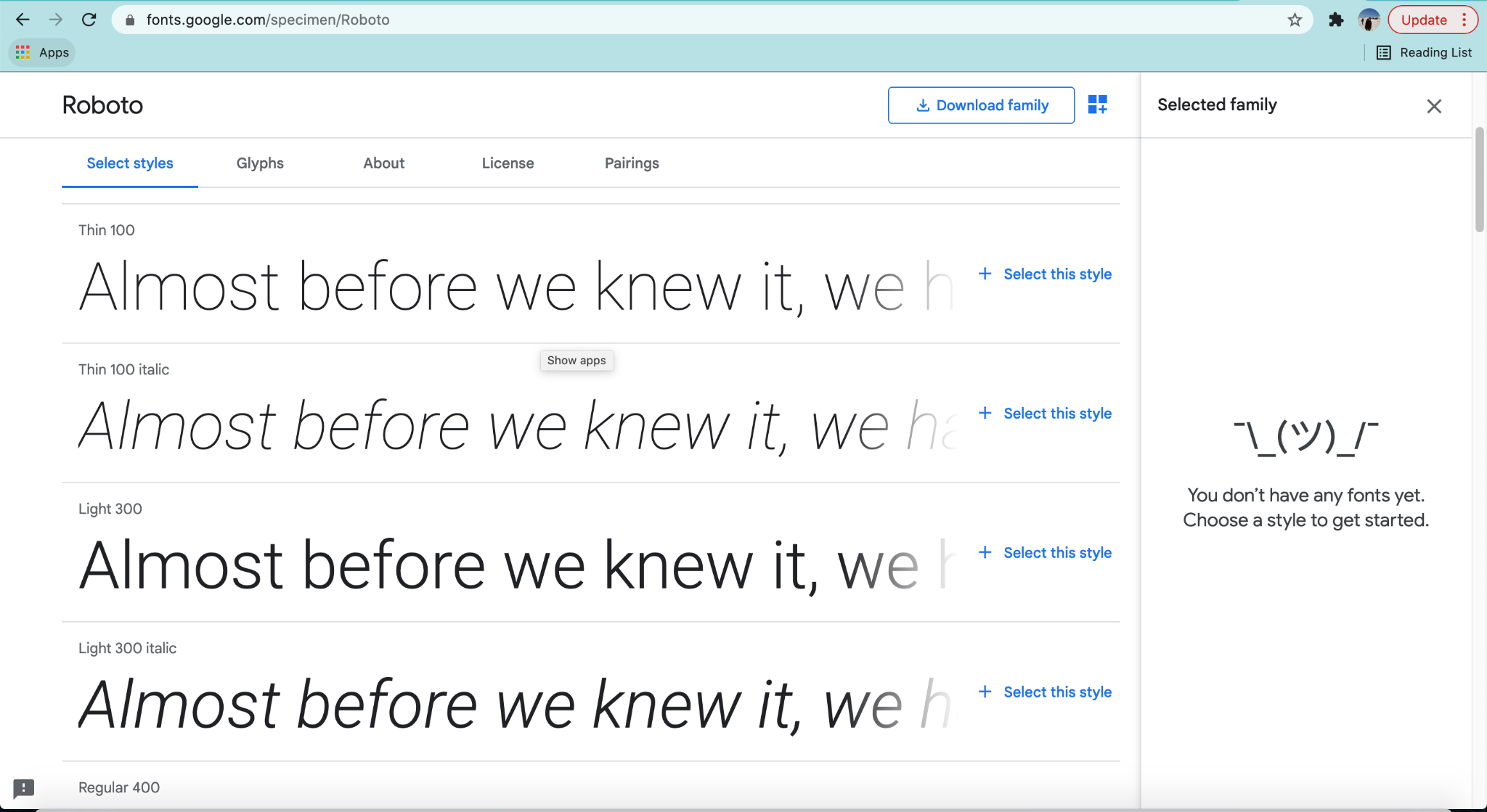
1. **Sticky** positioning. *Note that the HTML element with .position-sticky class is outside of the article. This is because sticky positioned elements can be affected only by scrolling on their direct parents.* :
   1. Set **sticky** position on .position-sticky class and scroll the page to see it work.
   2. Add a different background colour to the .position-sticky class
   3. Set a top **position** value to change where the element ‘sticks’ to

**IV. Styling text**

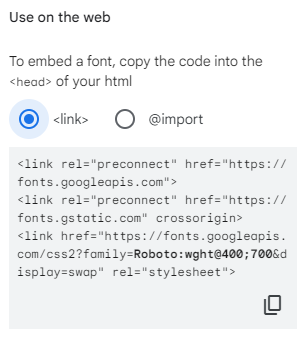
In this section we will focus on styling text. Search for the answers on the Internet.

1. **Color** - set a font color using the color CSS rule on the .text-style class
2. **Font size** – add a font-size CSS rule to the .text-style class. Experiment with different sizes and units.
3. **Line height** – add a line-height CSS rule to the .text-style class. Experiment with different values and units – also with omitting the units.
4. **Font weight** – add a font-weight CSS rule to the .text-style class. Experiment setting the font weight with both **keyword** and **number** values, e.g. bold and 800 (choose from 100, 200, 300, 400, 500, 600, 700, 800, 900)
5. **Importing** fonts and **font-family**
   1. Visit <https://fonts.google.com/> and find a font that you like
   2. Click ‘Select this style’ on several variations of the same font (see screenshot below). Selecting font styles is used to save the bandwidth for the user, if we do not intend to use certain styles of the font then we should not select them.

Select the styles, making sure to choose the font-weight you set in Q4. `Regular 400` on the list corresponds to the font used when no font-weight is set at all.



* 1. Select the <link> option from ‘Use on the web’ and copy the stylesheet code into the <head> section of index.html.



* 1. Copy the font-family CSS rule into your .style-text class.   
     The sans-serif part of that rule activates in the situation where the previous font could not be loaded, e.g. we misspelled ‘Roboto’ or there is no internet connection. In that case, the browser will load the default sans-serif font.

A screenshot of a phone

Description automatically generated

***TIP:*** *Have a look at the difference between some serif and sans-serif fonts*

1. **Text alignment** - add a text-align CSS rule to the .text-style class. Experiment with different values.
2. **Letter spacing** - add a letter-spacing CSS rule to the .text-style class. Experiment with different values using px units.
3. **Word spacing** - add a word-spacing CSS rule to the .text-style class. Experiment with different values using px units.
4. **Text transformation** - add a text-transform CSS rule to the .text-style class. Experiment with different values.
5. **Text decoration** - add a text-decoration CSS rule to the .text-style class. Experiment with different values.

**V. Styling links**

In this section we will be styling links in different states:

* a, a:link - link in its untouched form
* a:visited - link that has been visited
* a:hover - link that is hovered with a cursor
* a:active - link that has been “hovered” using keyboard
* a:focus - link that has been clicked but the cursor is still not released or the browser is navigating to another webpage

*Note that links come with a lot of styling from the browser. It is common to reset all those styles before developing CSS to simplify the styling process.*

1. a, a:link. Style all of the links in index.html in their basic (default) state by:
   1. Setting a font **color**
   2. Modifying the **font-size**, **font-weight**, and **text-decoration** for your links
2. a:hover, a:active, a:focus. Style all of these ‘active’ links differently to default links by:
   1. Overriding the **color** with a darker shade
   2. Using a different **text-decoration** value
3. a:visited. Many websites do not apply special styling to visited links. Style yours differently by:
   1. Overriding the **color**
   2. Using a different **font-style** value ([see this link](https://developer.mozilla.org/en-US/docs/Web/CSS/Privacy_and_the_:visited_selector) for why it doesn’t work!)

**VI. calc()**

In this section we will perform simple calculations using **calc**.

1. Add a new fixed width rule: **width: 100vw;** to the .calc-style class. The element should now overflow outside of the screen.
   1. Add a **background colour** to the .calc-style class to easily see its boundaries
   2. Add a new rule: width: calc(100vw - 100px); to the .calc-style class. Now the element occupies 100px less than the entire width of the screen.

*While in this example we could just use `width: 100%`, sometimes you will come across scenarios where the `width: 100%` solution does not work due to other CSS properties, or fixed width values. A lot of CSS frameworks use calc() internally so when we are using CSS frameworks we usually do not need to use calc().*

1. Create a new selector for the .fixed-width-col class to set its **width** to **350px** and **display: inline-block;**
2. Create a second selector for the .fluid-col class and set **display: inline-block;** on it as well. This will display the fixed-width-col stacked on top of the fluid-col.
   1. Use calc() to subtract the width of the fixed-width-col from the total page width and set a dynamically calculated **width** for the fluid-col so that the two elements appear side-by-side.