MORE SELECTORS & SPECIFICITY

INTRODUCTION

So far we have learnt the most basic ways to integrate CSS into our html pages. We have 3 different places to put our CSS:

- External (please use this one)
- Internal
- Inline

We also learnt a couple of different basic selectors to target things in our HTML:

- Class selector : .class-name
- Id selector: #id-name
- Tag selector: tag-name

In these slides we are going to learn about a few more selectors and what happens when you have CSS properties that conflict with each other.

MORE SELECTORS

While you can do a lot with the selectors we know so far, there are more kinds of selectors we can use for specific use cases.

Let’s just take a look at the different selectors available to us one at a time.

CHILD SELECTOR

Child selector: You can select **a direct child** by combining selectors with a > character.

```
selector>selector {

}
```

```
nav>a {
      . . .
}
```

So any **a** tag **directly** inside of a **nav** tag will be styled here.

ALSO SELECTOR

Also selector: You can make rules apply to multiple different selectors by using the , character.

```
selector,selector {

}
```

```
h1,h2,h3,.purple_text {
    . . .
}
```

Any rules in here will target h1 tags, h2 tags, h3 tags and any tags with the class .purple-text.

KNOWLEDGE CHECK

Let's start using some more specific selectors and see what kind of power this grants us:

1. Create a directory called CssSelectors inside Scratch
2. Follow the Git/Github workflow to turn this into a repository
3. Create a style.css file and link it in your index.html file
4. Add at least 5 content tags to index.html
    1. Make sure you nest some tags within others
5. Create a rule using the child selector at least 2 times
    1. Change whatever style you want
6. Create a rule using the also selector at least 2 times
    1. Change whatever style you want
7. Add, commit and push your code.

DESCENDANT SELECTOR

Descendant selector: You can select a **descendant** by simply combining selectors with a space character.

```
selector selector {

}
```

```
article .red-text {
    ...
}
```

So any class .red-text descending from an article will be selected by this rule. The .red-text class does not need to be a direct child of the article! It can be any descendant.

ADJACENT SELECTOR

Adjacent selector: You can select an adjacent element by combining selectors with a + character.

```
selector+selector {


}
```

```
#awesomeLogo+h1 {
      . . .
}
```

This selector is very specific. It is looking for the tag with the id of awesomeLogo, and then selecting the h1 tag that must be **directly after** it.

YOUNGER SIBLING SELECTOR

Younger sibling selector: You can select a younger sibling element by combining selectors with a ~ character.

```
selector~selector {

}

 .awesomeText~a {

 }
```

This selector will style all a tags that come **after** the .awesomeText class that also **share the same parent.**

KNOWLEDGE CHECK

Let's start using some more specific selectors and see what kind of power this grants us:

1. Add at least 5 more content tags to index.html
   1. Make sure you nest some tags within others
2. Give at least 3 tags a class and at least 3 tags a unique id
3. Create a rule using the a descendant selector at least 2 times
   1. Change whatever style you want
4. Create a rule using the adjacent selector at least 2 times
   1. Change whatever style you want
5. Create a rule using the younger sibling selector at least 2 times
   1. Change whatever style you want
6. Add, commit and push your code.

COMBINING MULTIPLE SELECTORS

With the more complicated selectors you can start doing some wild combinations to suite your exact needs for your page.

```
.awesomeText~a div>.verySpecific {


}
```

This selector will target all tags with the verySpecific class that are a child of a div. Those divs must be a descendant of an a tag that is a younger sibling of any tag with a class of awesomeText.

SPECIFICITY

The order of rules that your CSS is applied is called the "specificity". CSS rules are read in order from least precedence to most precedence overwriting as it goes.

From most Specificity to least, the order is the following:

1. The !important rules (you should avoid using this)
2. Inline style (the style attribute)
3. How specific the selector is (IDs vs classes vs HTML tags)
4. Code order (even the order in which css is linked)

It should be noted that when talking about specificity, things are pretty simple when talking about one selector but when combining selectors it can get a little bit more complicated.

SPECIFICITY CONT.

The !important tag is a last resort that should not be used. Needing to use this probably means that your CSS is poorly formed and needs to be redone.

The !important tag can be placed in your CSS rules to make them the most specific (it will not be overwritten).

Example:

```
p {
  color: red !important;
}
```

SPECIFICITY CONT.

Inline CSS has the second highest level of Specificity. We don’t want to write this in our HTML because it will make the code messy and hard to follow.

Later in the course when we start using JavaScript, we will start “injecting” CSS into our DOM. The CSS we inject will be at the inline level / specificity.

Using inline CSS can only be overwritten by using the !important override. For the most part we should really avoid doing inline CSS.

SPECIFICITY CONT.

What happens when you have a tag in your HTML that has multiple classes, and an ID?

You could have CSS rules clashing with each other. In this case there is a simple order:

- ID is most important
- Class is second
- HTML tag is last

So if you run into issues where a class or HTML tag rule is not showing, it might be because there is an ID rule overwriting it.

Again, this only really counts for using one selector. If you start combining selectors specificity gets a little more complicated.

SPECIFICITY CONT.

Your CSS rules are read by the browser in order from top to bottom. This also includes all of the CSS files you have linked to your page (the first one linked is the “highest”).

Being higher in order of being read means that your rules can be overwritten i.e. the last rules read will overwrite any rules above.

Note that the inline and more specific rules will still take precedence.

```
p {
    color: red;
}

p {
    color: blue;
}
```

While they might not seem necessary right now, I promise when you start working on your projects you will have situations where you will need to use different specific selectors.

Knowing specificity will save you a headache when debugging as well. You might think your CSS just isn’t being applied, but it could just be overwritten!