In this section we're going to discover the principles of CSS Architecture from the point of view of software engineering. We will learn the single responsibility principle, the open/closed principle, and separation of concerns from the perspective of writing maintainable CSS code. You will also discover what it's like to dry up your CSS, if it was to wet.

Normally I dislike quoting Wikipedia, but this time this resource summarized quite well what single responsibility principle is all about: every module or class should have responsibility over a single part of the functionality provided by the software. That responsibility should be encapsulated by the module or class.

The single responsibility principle is about one single responsibility that we have in our class, or module, or whatever unit in our CSS. And it is also about encapsulating that single responsibility inside, meaning that we're not gonna share this responsibility with other entities. It's encapsulated within the entity.

For instance, in the last lesson, we saw that we separated classes for styling, and classes for functionality. One class should have one single responsibility encapsulating whether it's going to be used in the CSS code for styling, or in the JavaScript code for functionality.

Whenever defining styling classes, it's important to note that it should serve one specific purpose. And if you break down these styling classes so that they serve one specific feature or aspect of styling, they're going to be reusable. This is how we can define for instance a hierarchical style sheet as well.

Another example for the single responsibility principle is that we can define modules inside our CSS code to shield that specific CSS from the outside world. So you can see that a single responsibility principle is very useful on all levels, on class level, as well as on module level.

Whenever we consider the maintainability of our code, it's very important to note that we have to formulate abstractions that are reusable. If we abstract too much, we're doing premature optimization once more because creating abstractions that no one ever uses is not too useful in CSS architecture. Whenever you can avoid ending up with an unmanageable number of classes attach to an element for instance because now you have a hard time maintaining your classes.

This is an example where we violated the single responsibility principle:

```html
```

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

☐

Single Responsibility Principle in CSS
The role of the Single Responsibility Principle in CSS architecture

```
```

```css
.posts-main { margin: 0 auto; width: 90%; }

.blogpost { width: calc( 33% - 16px ); background-color: #ccc; padding-bottom: 1rem; margin: 0; display: inline-block; border: 8px white solid; }

.featured-article { width: calc( 33% - 16px ); background-color: #7f7; padding-bottom: 1rem; margin: 0; display: inline-block; border: 8px #050 solid; }

.blogpost__thumbnail { width: 80%; margin: 10% 10% 1rem 10%; }

.blogpost__title { font-size: 1.4rem; margin: 0.25rem 10%; }

.blogpost__summary { margin: 0.25rem 10% 1rem 10%; }
```

There are six blogposts. Notice the `.blogpost` class and the `.featured-article` class. You can recognize quite a few similarities between the two because some of the rules just got copy pasted from one place to another. This is a violation of the single responsibility principle.

We can simply reuse the `.blogpost` class everywhere and then use the `.featured-article` class to describe the difference between the regular blogpost and the featured article. This implies that whenever we change the generic blogpost, we don't have to change the featured class anymore because of the reason that the feature article is also a blogpost and the generally cruel for blogpost applied for featured articles as well. As a side note we can implement it using the `@extend` feature of Sass.

```css
.posts-main { margin: 0 auto; width: 90%; }

.blogpost { width: calc( 33% - 16px ); padding-bottom: 1rem; margin: 0; display: inline-block; }

.blogpost--normal { /* @extend .blogpost; */ background-color: #ccc; border: 8px white solid; }

.blogpost--featured { /* @extend .blogpost; */ background-color: #7f7; border: 8px #050 solid; }

.featured-article { width: calc( 33% - 16px ); background-color: #7f7; padding-bottom: 1rem; margin: 0; display: inline-block; border: 8px #050 solid; }

.blogpost__thumbnail { width: 80%; margin: 10% 10% 1rem 10%; }

.blogpost__title { font-size: 1.4rem; margin: 0.25rem 10%; }

.blogpost__summary { margin: 0.25rem 10% 1rem 10%; }
```

This wraps up the section on single responsibility principle. You can read a lot more about it on the Internet.