Descriptive naming is one of the top creative challenges that software developers face. Whenever we name a class, we have to find the shortest possible class name. To make matters even worse, the usage of CSS classes is somewhat overloaded. This is because we can use a CSS class for styling purposes and also for application logic.

Therefore, it is very important to learn the ins and outs of naming CSS classes in a way that your application stays maintainable. We will start with an important concept in class naming, the separation of classes used for styling and classes used for functionality, in other words, JavaScript tools.

Whenever we touch a common resource used for styling as well as for functionality, we increase the probability of making a mistake. Both things may ruin the work of the other team. Common classes act as a minefield that may explode at any time. This is why we prefer separating classes for styling and classes for functionality.

According to the literature, we mostly use the `js-` prefix for JavaScript hooks. All the rest of the classes are named without the `js-` prefix. This way we keep the two types of classes separated and even independent teams can work on styling and application logic without interfering with each other.

Let's continue our discovery of proper class naming with exploring the possibilities of naming classes used for styling. The most widespread naming convention is the **block-element-modifier syntax**. The most comprehensive tutorial on this topic is en.bem.info.

Suppose we have a form acts as a block in our application. Each and every block in our application may contain elements. For instance, an account form may contain rows. We symbolize that the row belongs to an account form with the class name `.account-form__row`.

This is how we symbolize porthole relationships, you can see that the block is the account form and the row is an element inside the account form. And the relationship between the block and the element is joined by the two underscores in the block element syntax. Blocks are logical units in your CSS.

They act like a namespace in programming language or you can also call them a *scope*. Inside a block, you can use any names you want without clashing with another part of your CSS code. In addition, you can also style account form roles differently than, for instance, order form roles.

Another approach for viewing blocks is that they are reusable components in your page. This approach goes hand in hand with name spacing or scoping, as a reusable component style should be encapsulated into components. Always use descriptive names by selecting the shortest possible phrase for describing the purpose of the block.

Aren't we risking writing WET code? Imagine, if we had two different form rows, the class names would not make it possible to abstract common styles. Therefore we may run the risk of writing WET code which means, once more, that we enjoy typing. In practice, this risk doesn't really exist.

If you're using Sass or instance, you can abstract comment styles denoted by `.form __row`. Use `@extend` to mix these style properties in `.account-form__row` and `.order-form__row`. This makes it sufficient for you to just use the latter two class names in the markup.

Don't get carried away by nesting elements inside elements. Remember, blocks are for separating logical chunks of functionality. I doubt form rows are that significant in your code base that they deserve to go on block level.

By any chance if you wanted to symbolize the hierarchy, you could define a block account form row and just use `.account-form-row__text` field. This would also be sufficient, however an `.account-form__row` just doesn't seem to be a reusable component in my opinion that could be placed in your document individually. It can only be placed inside an `.account-form`. This is why our block is `.account-form` and `.account-form__row` is just an element inside the block.

One important rule is that you shouldn't add the modified and the unmodified version of the same block element or block to the same DOM node. These classes should be mutually exclusive. It is absolutely fine to use two modified versions of the same class though. For instance, you may indicate the same form field as mandatory and disabled at the same time.

BEM may be too much for you in case you create a simple website. Alternatively, even in a complex project, you may want to stick to different

conventions that suit your needs better. Object-oriented CSS is one option, according to the description you can scale the stylesheets of a thousand pages with it.

You may also use a systematic framework like Bootstrap. When using Bootstrap, you may have to tolerate its classes together with your own block-element-modifier classes. This is not a big problem though as a framework like Bootstrap is supposed to be common knowledge. When using the block-element-modifier syntax, consider some edge cases where we can track and drop or artificially reposition elements from one block to another