CSS Structure

Learning Objectives

- Understanding how the browser decides which CSS property to use when there are conflicting rules, regarding CSS cascade and CSS specificity
- Structuring CSS to improve maintainability and readability
- Organizing and naming CSS classes with BEM method
- Creating and reusing Custom CSS properties, also called CSS variables

---

CSS Cascade

The cascade is the algorithm that defines which CSS rules are being applied when there are conflicting rules.

When styling an element the browser:

1. Searches for all rules with matching selectors
2. Sorts the rules by their importance taking into account:
    - Whether the declaration is followed by **!important**
    - The rule's origin (Browser stylesheet, User stylesheet, Author stylesheet)
3. Sorts rules by their specificity, if there are multiple rules with the same importance according to no. 2.
4. Chooses the last declaration over previous ones, if there are multiple rules with the same importance and the same specificity.

You can read about the details in the CSS Cascade mdn docs.

> ❗ We recommend never using !important unless you absolutely have to. It is almost impossible to overwrite a CSS rule with !important.

Specificity

The specificity of a CSS selector tells the browser which rule is most relevant for an element. The more specific rules win over less specific ones.

You can find a list of the specificity of different selectors on specifishity.com.

The **universal selector** is the lease specific one. It is overwritten by any other CSS rule with any other matching CSS selector.

type selectors like `div` have a low specificity and can easily be overwritten.

class selectors like `.bright` and **attribute selectors** like `[type=checkbox]` have a higher specificity.

CSS Structure best practices

- keep your CSS consistent throughout a project. In collaborative projects, there are often coding style guidelines.
- separate global and local styles into different files (or sections of files)
- create multiple stylesheets for different parts of your application
    - structure your code by thinking in reusable **components**. You can write your CSS for every component in its own CSS file.

How to import one stylesheet or multiple stylesheets into another stylesheet

You can import one stylesheet into another stylesheet using **@import**:

```
@import "customer-card.css";
```

BEM

BEM is short for "Block, Element, Modifier". It is a method that allows you to craft reusable components through CSS class naming conventions.

```
.block {
  ...;
}

.block__element {
  ...;
}

.block--modifier {
  ...;
}
```

A **block** is a standalone entity or component.

An **element** is a part of your block (or component) that has no standalone meaning.

A **modifier** is a flag on your block (or component) that is used to change its appearance or behavior. E.g. disabled, checked, bright, etc. .

You can find an introduction to BEM here.

Kebab Case naming convention

The kebab case naming convention defines to use hyphens to separate words in variables. Many developers use the kebab case convention to write css classes. In BEM we also use kebab case, for example:

```
.customer-card {
  ...;
```

```
}

.customer-card__button {
  ...;
}

.customer-card--disabled {
  ...;
}
```

Custom properties (CSS variables)

You can store values in custom properties, so you can use them again multiple times without having to write the value.

A common practice is to define variables in the `:root` pseudo class selector as follows:

```
:root {
  --primary-color: #ff00ff;
  --secondary-color: #f00f0f;
}
```

> ! Custom properties have to be prefixed with `--`

You can use the custom properties as follows:

```
.customer-card {
  color: var(--primary-color);
  background-color: var(--secondary-color);
}
```

Resources

- MDN docs: CSS Cascade
- specifishity.com
- Introduction to BEM