Springboard

CSS Selectors and Specificity

Css Selectors and Specificity Demo code.zip 3.8KB

Goals

- Compare and contrast element, id and class selectors
- Review more complex selectors
- Understand how specificity works

CSS Selectors

In CSS we have tons of options for selecting specific elements or groups of elements

We'll see some more differences between these selectors soon, for now let's get used to the syntax.

Select by the name of the element

The simplest selector we can use is the **element** selector.

```
body { height: 300px; }
```

Select by an ID

Another way we can select elements is by using the **id** attribute.

```
<section id="main-heading"> Only one main heading here!
</section>
```

```
#main-heading { height: 500px; }
```

Remember, id attributes should be unique!

Why would you use an id?

- To uniquely identify an element
- To be very specific - we'll see more about that later!

Select by a class

A more common way we can select multiple elements is by using the **class** attribute, which allows us to select multiple elements.

```
<section class="post-description"> Our first post! </section>
<section class="post-description"> Our second post! </section>
<section class="post-description"> Our third post! </section>
```

```
.post-description { height: 300px; }
```

Why would you use a class?

- Select many different kinds of elements
- Be a little more specific than just an element selector

A few more selectors

- descendant
- adjacent sibling
- direct children

descendant selector

Target all paragraph tags inside of the body, regardless of how much nesting exists

```
<body> <p>Paragraph 1</p> <p>Paragraph 2</p> <section>
<p>Paragraph 3</p> </section> </body>
```

```
body p { color: green; }
```

Read this from right to left

"All paragraphs **inside** of body elements"

adjacent sibling selector

Target only the paragraph tags that come after h2 tags.

They must have the same parent element as well.

```
<body> <p>Welcome back!</p> <h2>Hi there!</h2> <p>Paragraph
2</p> <section> <p>Paragraph 3</p> </section> </body>
```

```
h2 + p { color: green; }
```

direct children selector

Target only the h2 tags that are direct children of section tags

```
<body> <section> <h2> Yes me!! </h2> <article> <h2>Not me!<h2>
</article> </section> </body>
```

```
section > h2 { color: green; }
```

Multiple selectors

We avoid code like this:

```
h2 { color: blue; } .main-section { color: blue; } #top-
heading { color: blue; }
```

By doing this:

```
h2, .main-section, #top-heading { color: blue; }
```

Pseudo class

A CSS pseudo-class is a keyword added to a selector that specifies a special state of the selected element(s).

For example, :hover can be used to change a button's color when the user's mouse hovers over it.

```
/* Any button over which the user's mouse is hovering */
button:hover { color: blue; }
```

You can find more of them at https://developer.mozilla.org/en-US/docs/Web/CSS/Pseudo-classes

CSS Specificity

Specificity

Some selectors are far more specific than others!

Always remember the following order from least to most specific

1. element
2. class
3. id
4. inline style
5. !important

Inline styling

We can also just write our CSS directly in HTML elements using the ***style*** attribute

```
<section> <p style="color:red;">It's nice to see you!</p>
</section>
```

While this may seem convenient, it's very dangerous. Inline styling will always override IDs, classes and element selectors!

!important

Think of it as the **nuclear** option. You should almost never use it!

It will override everything and anything.

```
<section> <p style="color:red;">It's nice to see you!</p>
</section>
```

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

Even our inline style doesn't stand a chance.

What happens if they have the same specificity?

```
h1 { color: green; } h1 { color: blue; }
```

That's what the **Cascading** in CSS helps us with!

Whichever declaration comes **last** will be the selected on.

Recap

- Common CSS selectors include element, class, and id
- Be careful with inline styling because it is so specific
- Don't use !important