Lesson:

Basic Selector in CSS and their specificity and cascading order

PW SKILLS

Topics Covered

- Introduction to Selectors in CSS
- Categories of CSS Selectors
- Basic Selectors with Example
- Grouping selects with Example
- Combinators in CSS
- CSS Specificity and Cascading order

Introduction to Selectors in CSS

CSS (Cascading Style Sheets) selectors are used to targeting specific HTML elements or groups of elements in order to apply styles to them. CSS selectors allow us to specify which HTML elements we want to style, based on their attributes, classes, ids, and other characteristics.

Categories of CSS Selectors

It can be grouped into the following categories based on the type of elements it can select.

They are -

1. Basic selectors
2. Grouping selectors
3. Combinators
4. Pseudo-classes and pseudo-elements.

Basic Selectors with Example

The basic selectors include the following - Universal selector, Element selector, Class selector, ID selector, and Attribute selector.

Universal selector -

The universal selector in CSS is denoted by an asterisk (*) and it targets all elements on a web page. It can be used to apply styles to every HTML element within a specific scope, such as a specific container or the entire document.

style.css

```
Unset
/* Universal selector syntax */
* {
   ...styles properties
}

/* Example of Universal Selector*/
* {
     margin: 0;
     padding: 0
}
/*
     This will remove all the default margin and padding size
to the HTML document.
*/
```

Element selector -

An element selector in CSS targets HTML elements based on their tag name. It allows you to apply styles to all occurrences of a particular HTML element throughout your web page.

style.css

```
Unset
/* Element selector syntax */
tagName {
   ...style properties
}

/* Example of Element Selector*/
h1 {
      color: red;
}
/*
      Change all the h1 element color to red.
*/
```

Class selector -

A class selector in CSS targets HTML elements based on their class attribute value or class name. It allows you to apply styles to one or more elements that share the same class name.

index.html

```
Unset
<!-- Demo HTML code -->
<button class="cta-btn"> Click Me </button>
```

style.css

```
Unset
/* Class selector */
className {
   ...styles properties
}

/* Example of Class selector */
/* using class name of the html code */
.cta-btn {
  border: 1px solid black;
}

/*
     This will give one 1px of the black solid border to the
button element with a class name of `cta-btn`
* /
```

Dote (.) followed by the class name is used to select the particular HTML element.

ID selector

An ID selector in CSS targets a specific HTML element based on its unique ID attribute value. It allows you to apply styles to a single element with a unique ID. Hash(#) followed by the id name selects the particular HTML element.

index.html

```
Unset
<--! demo html file -->
<div id="hero-section"> ...</div>
```

style.css

```
Unset
/* ID selector syntax*/
#idName {
 ...styles properties
}

/* ID selector example code*/
#hero-section {
     background-color: blue; }
/*
This will change the background color of an element with id
"hero-section" to blue
*/
```

Attribute selector

An attribute selector in CSS is used to target HTML elements based on the presence, value, or partial value of an attribute. It allows you to apply styles to elements with a specific attribute, or elements whose attribute values match a particular pattern.

index.html

```
Unset
<a href="https://inueron.ai>inueron.com</a>
<a href="https://pwskils.com target="_blank" >pwskills.com</a>
```

style.css

```
Unset
/* Attribute selector syntax*/
htmlTagName[htmlAttribute]{
 ...styles properties
}
/* Attribute selector example code*/
a[target]{
     color: red;
}
/*
This will change the anchor tag having attribute "target", to
red.
*/
```

Grouping Selectors with Example

In CSS, grouping selectors allow you to apply styles to multiple elements with a single rule, reducing the amount of repetitive code. Grouping Selectors involves separating individual selectors with commas (“,”). Any styles specified within the rule will be applied to all elements that match any of the group selectors.

Example -

```
Unset
/* Syntax of Grouping selector*/
element1, element2, element3 {
     ...styles properties
}

/* Example of CSS Group selector*/
h1, h2, h3 {
     color: red;
     font-size: 20px
     text-decoration: none
}
/*
This will change the h1, h2, and h3 to the following CSS
styles properties given.
*/
```

Combinators with Example

In CSS, combinators are used to select elements based on their **relationship to other elements** in the HTML document. Here are some of the different types of combinators in CSS, they are - Descendant combinator, child combinator, General sibling combinator, and Adjacent sibling combinator.

Descendant combinator

The descendant combinator (" ") in CSS is used to select an element that is a descendant of another element, regardless of how deeply nested it is in the HTML structure. The descendant combinator is represented by a space between two selectors in a CSS rule

index.html

```
Unset
<ul>
    <li> home</li>
    <li> about</li>
    <li> contact</li>
 </ul>
```

style.css

```
Unset
/* Syntax of Descendant Selector*/
selector1 selector2{
 ...CSS code
}
/* Example of Descendant Selector*/
ul li {
 color: red
}

/* This will change the color of list elements <li>. */
```

Child combinator

The child combinator (>) is placed between two CSS selectors. It matches only those elements matched by the second selector that are direct children of elements matched by the first. Elements matched by the second selector must be immediate children of the elements matched by the first selector.

index.html

```
Unset
 <header>
        <a href="#"> logo </a>
        <nav>
            <ul>
                ....
            </ul>
<!-- this anchor tag will not be changed since it is not an immediate child of
the second selector  -->
             <a href="#"> logo </a>
        </nav>
</header>
```

style.css

```
Unset
/*Child combinator syntax */
htmlElement1 > htmlElement2 {
 ...CSS code
}
/* Example of Child Combinator*/
header > a {
     color: aqua;
}

/*
     This will change the header immediate child anchor tag
color to aqua
*/
```

General sibling combinator

The general sibling combinator (`~`) in CSS is used to select all elements that come after another element with the same parent. The general sibling combinator is represented by a tilde (`~`) between two selectors in a CSS rule.

index.html

```
Unset
<div>
<img src="#" alt="demo" />
<p>Change me</p>
</div>
```

style.css

```
Unset
/*Child combinator syntax */
former_element ~ target_element {
 ... style properties
}
/* Example of Child Combinator*/
img ~ p {
      color: aqua;
}

/*
      Change the color of any paragraph to aqua that is a
sibling  to the image element
*/
```

Adjacent sibling combinator

The adjacent sibling combinator (+) separates two selectors and matches the second element only if it immediately follows the first element, and both are children of the same parent element.

index.html

```
Unset
<div>
   <img src="#" alt="demo" />
   <p>Change color</p>
</div>
```

style.css

```
Unset
/* Adjacent sibling combinator syntax */
former_element + target_element {
      ...style properties
}

/* Example of Adjacent sibling combinator syntax*/
img + p {
      font-weight: bold;
}
/*
      Change the paragraphs that come immediately after any
image to bold font-weight.
*/
```

Note - Pseudo-selectors are advanced levels of Selectors in CSS, it will be discussed as a separate topic in the next section.

CSS specificity and Cascading order

CSS specificity and cascading order are important concepts that determine how conflicting styles are applied to HTML elements when multiple style rules are applied.

CSS Specificity:
CSS specificity refers to the level of importance or weight assigned to a particular style rule. It determines which style rule takes precedence when multiple rules target the same HTML element. CSS specificity is calculated based on the combination of different selectors used in a style rule. The higher the specificity of a style rule, the more precedence it has over other rules

The following rules are used to calculate CSS specificity:

1. **Inline styles:** Inline styles are applied directly to HTML elements using the style attribute. They have the highest specificity because they are applied directly to the element and override other styles.

```
Unset
/*Example of inline style*/
<div style="color: red;"> Hello </div>
```

2. **ID selectors:** ID selectors are used to target elements with specific ID attributes. They have higher specificity than class selectors and element selectors.

```
Unset
/* Example of ID selector */
#myId { color: red}
```

3. **Class selectors :** Class selectors are used to target elements with a specific class attribute. They have lower specificity than ID selectors but higher specificity than element selectors.

```
Unset
/* Example of class selector*/
.classname{ color:black}
```

4. **Element selectors :** Element selectors target elements based on their tag names. They have the lowest specificity and are overridden by ID selectors, class selectors, and inline styles.

```
Unset
/* Example of element selector */
h1 { color: purple}
```

How to Calculate Specificity?
CSS specificity is calculated based on the type of selector used and is represented by a four-digit value.

Here is how to calculate specificity -

- Inline style gets a specificity value of 1000 and is always given the highest priority.
- Type selector, for each type selector, 1 is added to the specificity value.
- Class selector, for each class selector, 10 is added to the specificity value.
- Id selector, for each id selector, 100 is added to the specificity value.
- Universal selector is ignored, i.e. 0 is added to the specificity value.

The table below shows some examples on how to calculate specificity values:

SELECTOR	SPECIFICITY VALUE	CALCULATION
p	1	1
p.test	11	1 + 10
p#demo	101	1 + 100
<p style="color:pink;">	1000	1000
#demo	100	100
.test	10	10
p.test1.test2	21	1 + 10 + 10
#navbar p#demo	201	100 + 1 + 100
*	0	0 (the universal selector is ignored)

Note - The selector with the highest specificity value will win and take effect!
There is one exception to this rule: if you use the !important rule, it wil even override inline styles.

Example -
index.html

```
Unset
<body>
    <h1 class="heading" style="color: green;">CSS
Specificity</h1>
</body>
```

style.css

```
Unset
h1 {
  color: red;
}
h1#heading {
  color: yellow;
}
```

Browser output –

From the above code we get three specificities i.e –

1. h1 (type selector) with specificity value of 1
2. h1#heading (type selector with id) with specificity value of 100 + 1 = 101
3. Inline styling with specificity of 1000

Since the third rule with inline styling has the highest specificity value of 1000, this style declaration will be applied.

CSS cascading order
It refers to the order in which style rules are applied when there are multiple rules targeting the same HTML with the same specificity. The order in which the style rules are defined in the CSS file determines their cascading order.

The later rule in the CSS file takes precedence over the earlier rule when both rules have the same specificity. This means that if two rules with the same specificity conflict, the one defined later in the CSS file will be applied.

Example –

```
Unset
.heading {
  color: green;
  text-decoration: underline;
}
/* According to the Cascading order rule - the below rule will
be applied */
.heading {
  color: red;
  text-decoration: line-through;
}
```

Browser output –

"!important" Rule in CSS

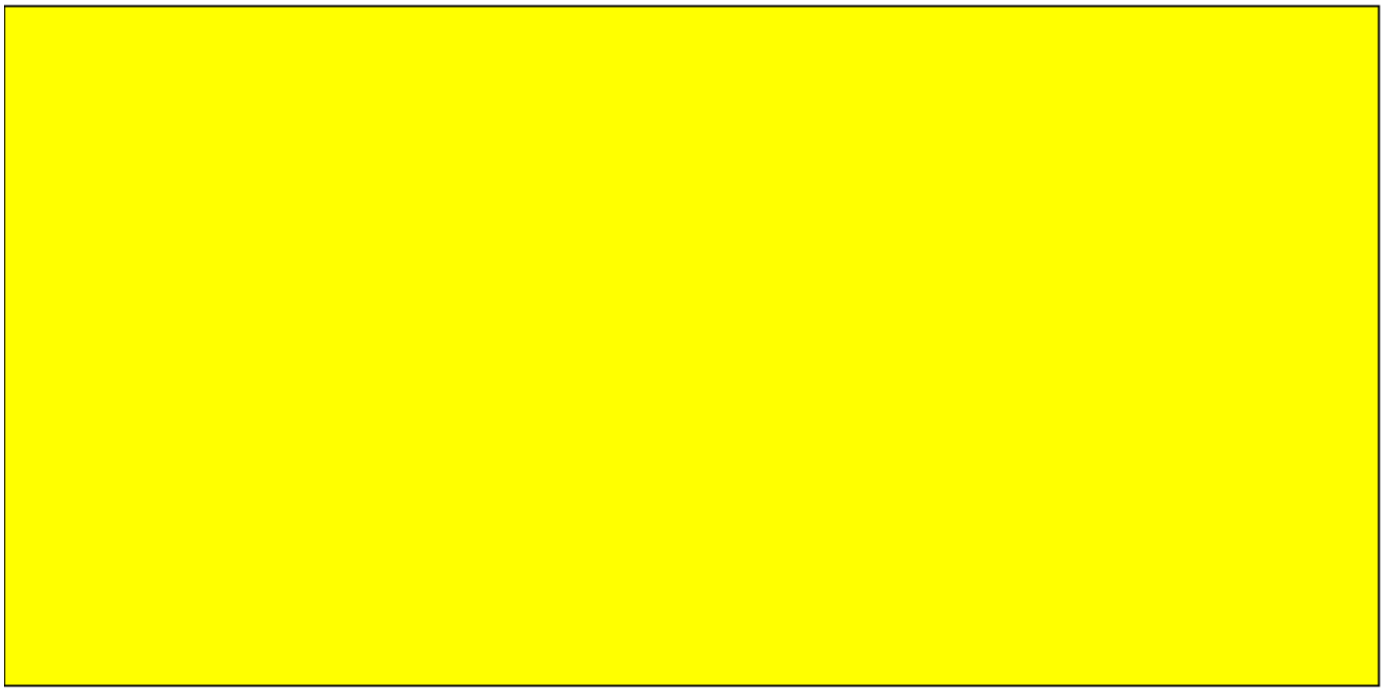
The "!important" rule is used to give style rule the highest priority, which means it overrides all other styles applied to the same element.

When the "!important" rule is added to a style property, it will take precedence over any other styles applied to the same element, regardless of where those styles are defined in the CSS file or in the HTML code. This can be useful in situations where a specific style needs to take precedence over all others, such as when debugging a layout issue or when working with third-party CSS libraries.

Example -

```
Unset
.container {
  height: 50vh;
  border: 1px solid;
  background-color: green;
}
/* This will override the background color*/
.container {
  background-color: yellow !important;
}
```

Browser output -



Advantages and Disadvantages of "!important" -

ADVANTAGES	DISADVANTAGES
It has the highest Priority, which means it overrides all other styles applied to the same element.	It is more difficult to maintain and update styles in the future. Because it takes precedence over other styles.
It is Easy to use since it is a quick and easy way to apply a style that overrides other styles in a particular context	It can create specificity conflicts and make it harder to override styles in specific situations
It improves accessibility by ensuring that a specific style is applied that makes content easier to read or use.	Debugging issues that arise from the use of **"!important"** can be more difficult, as styles can be overridden unexpectedly and cause layout issues.

When you have to use and avoid "!important" CSS rule

When to use -

- Overriding styles from third-party libraries: Sometimes, you may need to override styles from a third-party library that you cannot modify directly. In this case, using **!important** can be a quick and easy way to override the styles without modifying the original library.
- Debugging layout issues: In some cases, using **!important** can help you quickly identify and fix layout issues that are caused by conflicting styles.
- Applying accessibility styles: Using **!important** can help ensure that accessibility styles are applied consistently across your site, even if other styles conflict with them.

When to avoid -

- Maintenance and scalability: Overuse of the **!important** rule can make it harder to maintain and scale your styles over time, especially when working on larger projects.
- Specificity conflicts: Overuse of the **!important** rule can lead to specificity conflicts, making it harder to override styles in specific situations.
- Unpredictable behaviour: Overuse of the **!important** rule can lead to unpredictable behaviour in your styles, especially when working with larger and more complex stylesheets.