COMP 1017

Day 12
More CSS Selectors

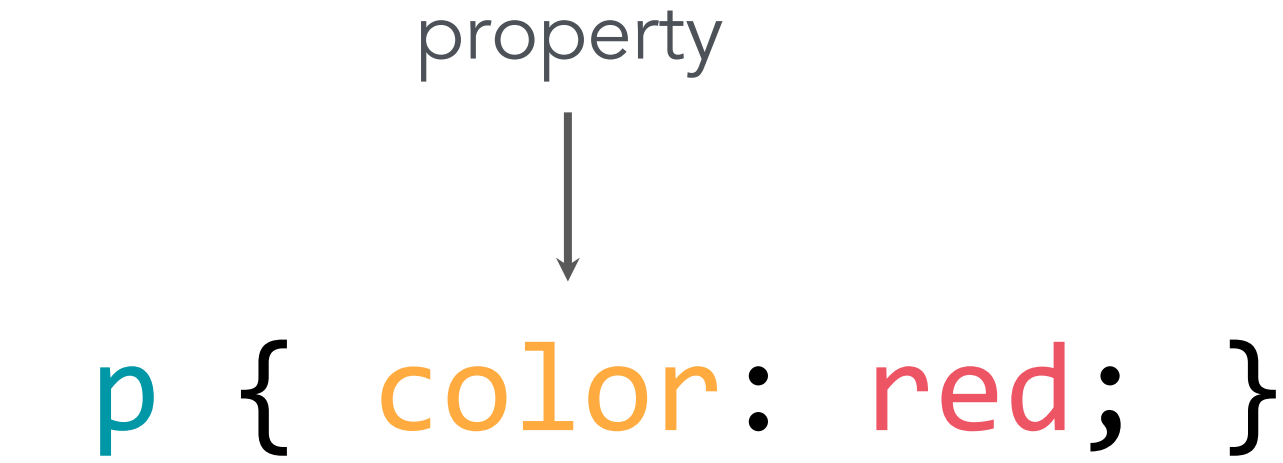
Let’s do a quick review of everything we’ve covered about CSS so far!

selector

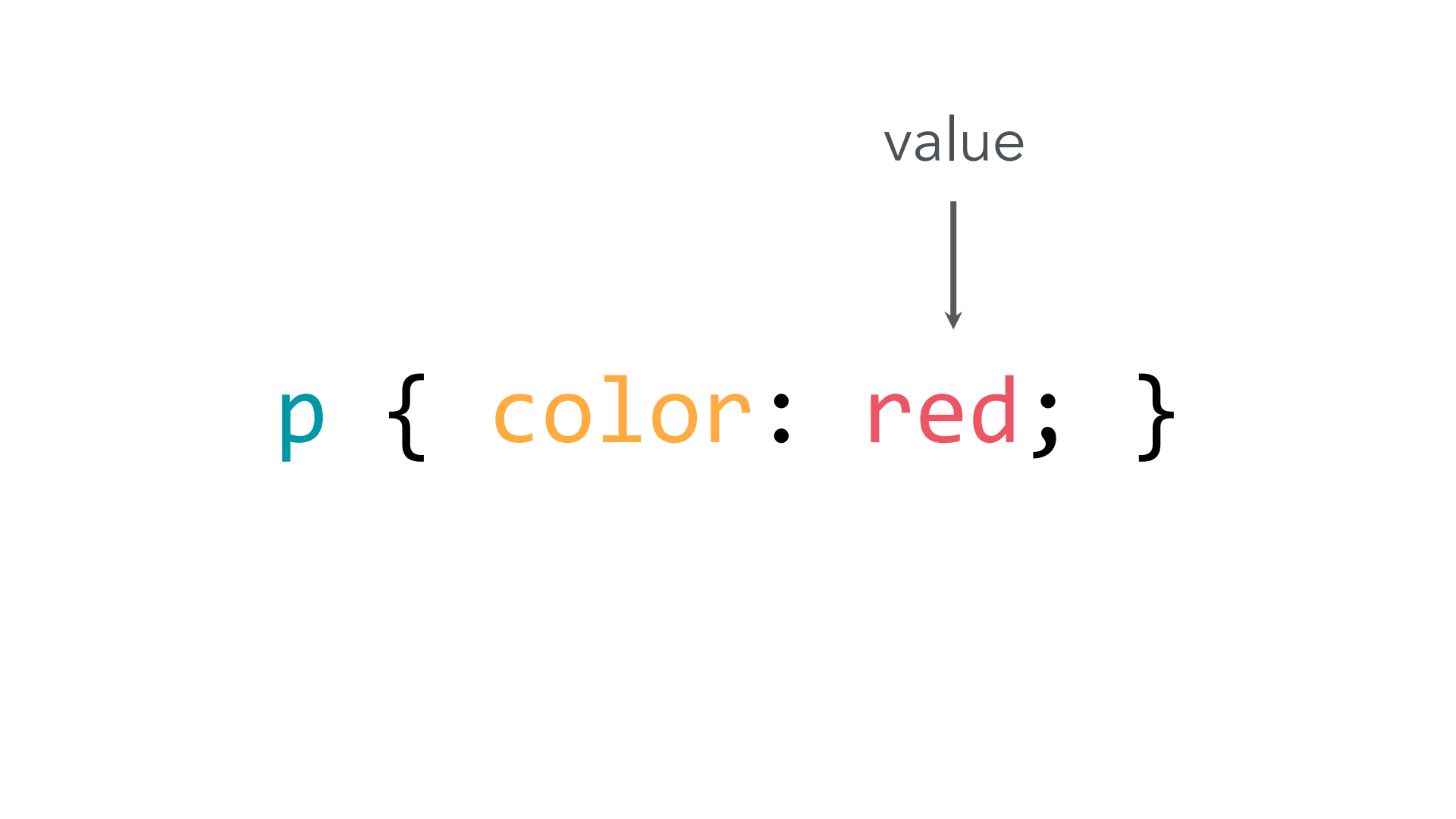
selector

```
p { color: red; }
```

property


property
p { color: red; }

value


value
p { color: red; }

declaration

```
p { color: red; }
```

Declaration (property: value;)

External (linked) stylesheet

Linked to the css file (styles.css) in the css folder

<head>

Inline & Embedded stylesheets

These are under the DO NOT USE ones!

the cascade

Last rule wins!!! Sort of…

.class

The class selector selects html elements with a specific class attribute…I don’t feel so lonely anymore!

CSS Selectors

There’s a few of these bad boys to discover!

So far, we've looked at element selectors and class selectors.

These are known as **simple selectors**.

You can add #ID in there too, but we won't be using them.

element selector

```
p {
    color: red;
}
```

class selector

```
.container {
    width: 960px;
}
```

Let’s look at a few more…

multiple element selector

Sometimes known as the grouping selector.

We can select multiple elements at a time, allowing us to apply the same rules to each element.

All it takes is a comma (,) between each element that we want to select.

multiple element selector

```
h1, h2 {
  color: blue;
}
```

/* Add a comma and a space between elements */

…we can also target specific elements **with** specific classes applied to them.

element class selectors

```
p.red-text {
  font-size: 56px;
}
```

```
/* Selects only <p> elements with the class
.red-text applied to it (no spaces between
them – read left to right). */
```

```
/* It will not change other elements with
   the class .red-text applied to it.
It will also not change the rules of the
            .red-text class. */
```

descendant selector

A descendant selector looks at an element's 'lineage' and place within the DOM.

It's that parent child relationship.

This allows us to be much more explicit about which elements we want to target.

It is written with a space between each element.

It starts with the parent element
and works inward.

Descendant Selector

```
header h2 {
  color: blue;
}
```

```
/* Selects only the <h2> elements inside the
<header> element. (space between them – read
right to left). */
```

Descendant Selector

```
footer a {
  text-decoration: none;
}

/* This only targets hyperlinks <a>
   inside of the footer. */
```

descendant combinator

A descendant combinator takes all of these things and puts them together.

element class selector + descendant selector

Descendant Combinator

```
ul.my-things li {
  color: blue;
}
```

```
/* This targets the <li> inside of any
  <ul> with the class of.my-things */
```

```
<ul class="my-things">
      <li>One</li>
      <li>Two</li>
</ul>

    <!-- The list items <li> are
         targeted here. -->
```

```
<ul>
      <li>One</li>
      <li>Two</li>
</ul>

 <!-- These are not as they do not have
    the class associated to them. -->
```

```
<ol>
      <li>One</li>
      <li class="mythings">Two</li>
</ol>

<!-- These are also not targeted as
   they are wrapped in a <ol> -->
```

pseudo-**class** selectors

A pseudo-**class** selector targets an element when it's in a certain state.

By now you might be in some sort of panic mode, but we're not talking about that sort of state!

One of the best examples of this is applying rules to hyperlinks to change their look or behaviour.

```
a:hover {
    text-decoration: underline;
}

/* When the user moves their mouse over the
<a> tag hyperlink, an underline will appear.
*/
```

CSS Battle Royale

When CSS rules collide,
who remains?

What's the first rule of our CSS Battle Royale?

The cascade determines the winner.

This means that when there are two conflicting rules, the rule written last will be rendered.

The last rule written is the last rule standing.

What's the second rule of our CSS Battle Royale?

The specificity of a selector determines the winner.

Remember we associated a numeric value to different types of selectors?

Let’s take a quick look at how specificity is calculated.

Selector & Example	Calculated Weight
Element Selector `p, h1, h2 …`	1
Class Selector `.container`	10
Descendant Selector `ul li a` (<a> tag inside of a in an)	2
Descendant Combinator `ul.my-things li` (inside of an with the class of .my-things)	11
ID Selector `#jumbotron`	100
!important `p { color: red !important; }`	101

The specificity of a selector can override the order of the cascade.

Introducing … rule number three!

In CSS we have a special piece of syntax that we use to make sure a certain declaration will win over all others.

It looks like this:

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

!important is the heaviest hitter of them all.
It's even stronger than #IDs.

In general, we do not recommend using `!important`*.

* or, to only use it *very sparingly*

But, if you must …

Always look for a way to use specificity before even considering `!important`.

Inheritance

This just might have something to do with parent-child elements …

Inheritance is one of the last pieces we need* to understand what style is applied to which element.

* well, for now …

Inheritance means that some property values applied to an element will also be applied to its children.

So, if you make the entire <body> use comic sans, then everything inside the <body> will look like poorly-kerned garbage.

For more on inheritance, check out this supplemental reading:

MDN: Cascade & Inheritance

For more on how to calculate specificity values, go see:

css-tricks.com/specifics-on-css-specificity/