CSS3

Q What is SEO? *4-6-1*

A *4-6-2*

chapter 5 / 7

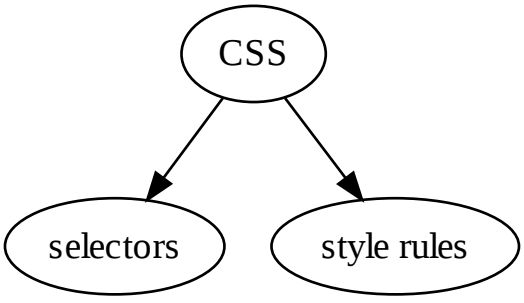
CSS3

Sections in this chapter:

5-1. CSS basics

CSS, or **Cascading Style Sheets**, is a language we use to control **style** and **layout** of HTML content. *5-1-1*

It is expressed through **selectors** and **style rules** which define how a certain element should be rendered in an HTML-page. *5-1-2*



All elements matching the selector will get the style rule. 5-1-3

```
selector {
    style-rule;
}
```

The **selector** specifies which elements to style and could for instance be an explicit HTML tag name, an ID or a class name. 5-1-4

```
body {
    style-rule;
}
```

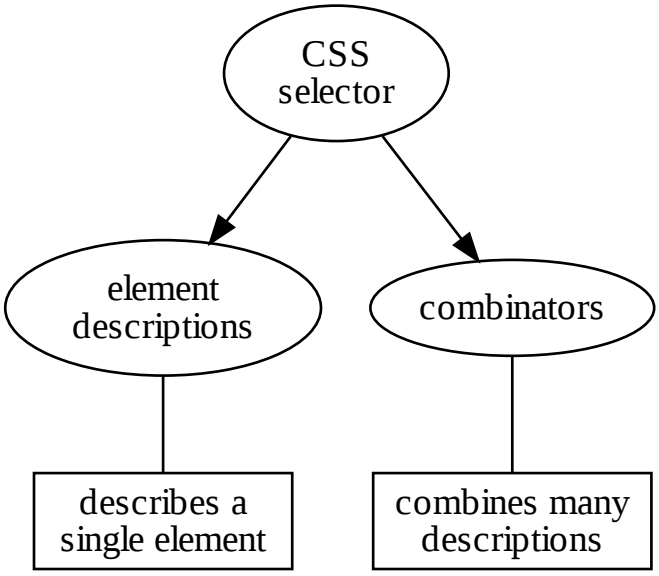
A **declaration** is written as a **property/value pair**, defining how to style the selected element. The property is an attribute, for instance `color`. The value depends on the attribute: 5-1-5

```
body {
    color: red;
}
```

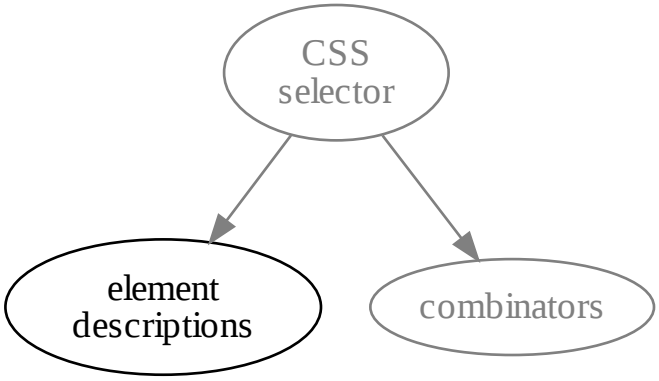
5-2. Selectors

To apply a CSS rule to an HTML element, we need to target the element using a selector. 5-2-1

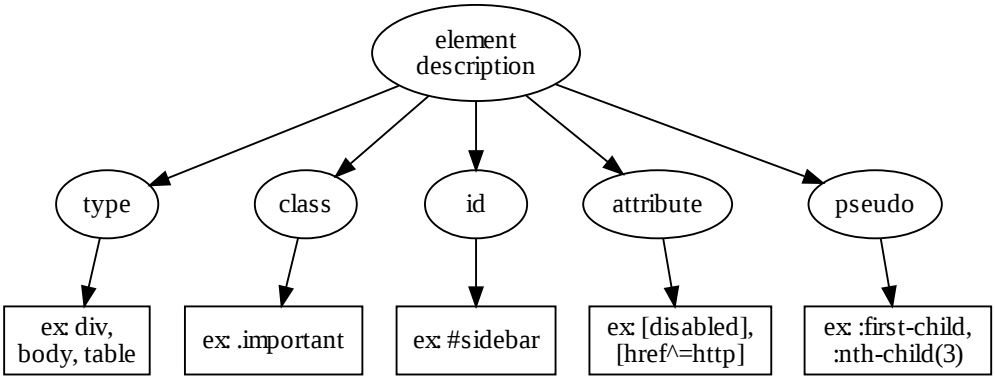
Just like CSS splits into **selectors** and **style rules**, so does **selectors** split into **descriptions** and **combinators**. 5-2-2



The most important part of a CSS selector is to **describe elements** that should be selected. 5-2-3



There are **five different aspects** that we can describe, each with its own syntax: 5-2-4



These **can be combined** however you see fit. Here is an (exaggerated) example using all of them: *5-2-5*

```
button[disabled]#deletemsg.big:first-child
```

This would match all *5-2-6*

- nodes of type **button**
- that has a disabled **attribute**
- and **id** is `deletemsg`
- and class attribute contains `big`
- and it is the **first child** of its parent

As per usual, **the details can be found on MDN**: *5-2-7*
https://developer.mozilla.org/en-US/docs/Glossary/CSS_Selector

We will, however, look at a few simple examples.

Let's say we have the following HTML: *5-2-8*

```
<h2>Course objectives</h1>
<p id="first-paragraph">
    To endow participants with awesome powers
</p>
<h2>CSS syntax</h2>
<p class="warning">Is awesome</p>
<a href="https://css-tricks.com/css-is-awesome/">
    CSS-Tricks
</a>
```

To select by type description: *5-2-9*

```
h2 {
    color:red;
}
```

The above rule will apply to all h2 headers, since the rule is set to all h2 elements.

To select by id description: *5-2-10*

```
#first-paragraph {
    font-size: 12pt;
}
```

This will apply the above rule to the first paragraph, with the id `first-paragraph`. An **id** uniquely identifies an element on a page, and may only be used for one element, not several.

To select by class: *5-2-11*

```
.warning {
    color: orange;
}
```

The above rule will apply only to the paragraphs with the class `warning`, in our case the second paragraph. A **class** is any name inside an HTML `class` attribute. It may be applied to several elements.

To select by attribute: *5-2-12*

```
a[href^="https"] {
    color: green;
}
```

There is also a selector which targets values in attributes that end with a specific word or value. *5-2-13*

```
a[href$="shop"] {
    background: green;
}
```

What if I want to target an element whose value contains a specific word or value? For this there is a contains selector: *5-2-14*

```
a[href*="products"] {
    background: green;
}
```

There is a bunch of ways to match values, check out MDN Attribute Selectors for more examples! 5-2-15

Q Ok, so ^= allows us to match against the **beginning of an attribute value**. But what would be the point of that first example we saw? 5-2-16

A If all local links are relative, which is normally the case, then this would be an excellent way to **catch all external links**, to make them look or behave differently. 5-2-17

Finally, some notes on the pseudo-classes: 5-2-18

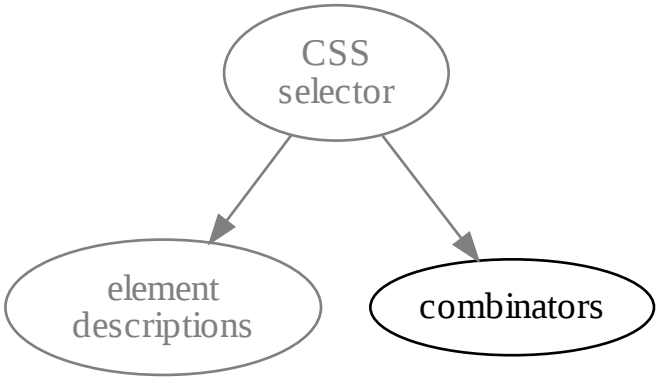
They are **prefixed with :**, and allow matching on **position or state**.

- `:first-child`
- `:link`
- `:hover`, `:active`, `:focus`

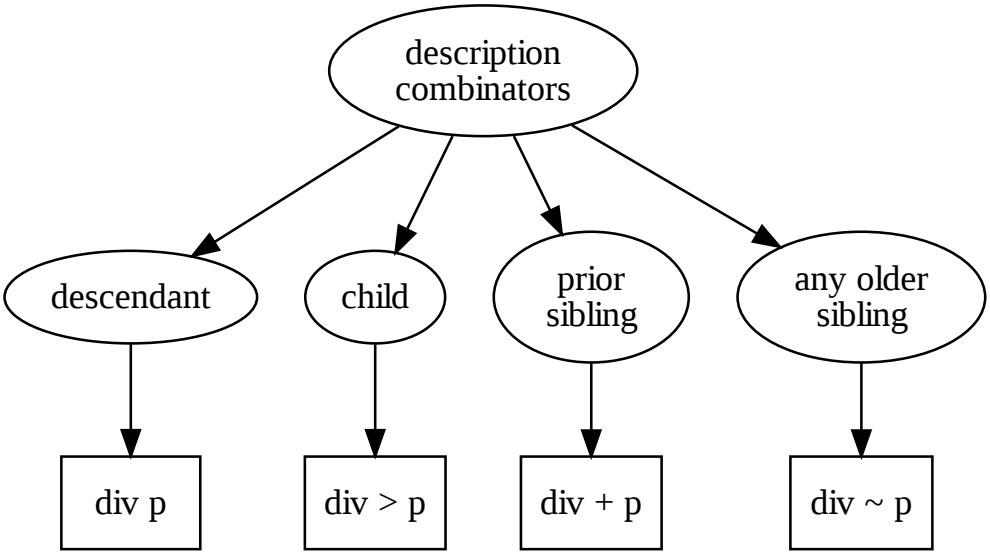
MDN has very good documentation about pseudo-classes.

5-3. Combinators

Let's now look at the other half of selectors, namely how we **combine** descriptions! 5-3-1



There are **four different ways** that descriptions can be combined, which we'll look at one at a time: 5-3-2



The perhaps most common one is the **descendant combinator**. By having **two descriptions with space between**: 5-3-3

```
div p
```

we match all elements that 5-3-4

- **match the last description**
- have an **ancestor matching the first description**. This can be any number of generations up the tree.

The **child combinator**: 5-3-5

```
div > p
```

is very similar to descendant selector, but here the first selector must match the **parent** and not just any ancestor.

Thus the child combinator is **smaller in scope** than the descendant combinator. 5-3-6

The **sibling combinator**: *5-3-7*

```
div ~ p
```

is similar to the descendant combinator, but **works horizontally** instead.

It matches elements that: *5-3-8*

- **match the last description**
- have an **older sibling that matches the first description**

Finally the **adjacent sibling combinator**: *5-3-9*

```
div + p
```

works in the exact same way, but requires the **neighbouring older sibling to match the first description**.

The two **sibling combinators are not often used**, but they are **good at what they do**.

Solving that problem with other means would require **brittle workarounds**, something you often see from web developers who don't know about them.

5-4. Inclusion

There are **3 different ways** in which we can **apply CSS to our content**: *5-4-1*

- linking to a **separate `.css` file**
- putting it into a `style` **tag**
- inlining it in the `style` **attribute** of an element

A separate file means using a `<link>` element: 5-4-2

```
<!DOCTYPE html>
<html>
    <head>
        <link href="style.css" rel="stylesheet">
    </head>
    <body>
        <!-- lots of HTML content here -->
    </body>
</html>
```

The `<style>` element simply **wraps the CSS code**. It can go anywhere, but customarily in the head: 5-4-3

```
<!DOCTYPE html>
<html>
    <head>
        <style>
        /* Lots of CSS code here */
        </style>
    </head>
    <body>
        <!-- Lots of HTML content here -->
    </body>
</html>
```

Inlining in the `style` **attribute** means we don't need selectors, since the style rules are applied to this particular element. 5-4-4

```
<p style="font-weight: bold;">It will never happen again</p>
```

Q Which method do you think is the most common, and why? 5-4-5

A In almost all situations **a separate file is preferred**, since that gives us a good separation of concerns. Style tags arguably gives us that too, so the primary thing is to **be careful with inlining styles**. 5-4-6

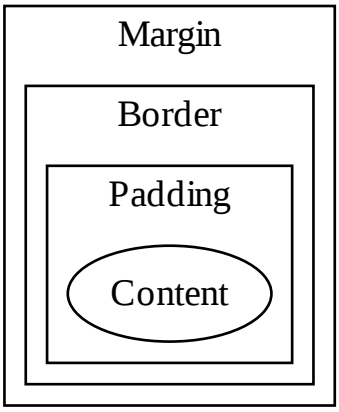
5-5. Box model

Elements rendered on a page are rendered in rectangular boxes. In the middle is the **content area**. It's surrounded by optional **padding**, **border** and **margin** areas. *5-5-1*

Padding, **border** and **margin** are used to add spaces between items in a page. *5-5-2*

The edge around the content area is sometimes called the **inner edge**, and the one around the margin the **outer edge**. *5-5-3*

5-5-4



The **content area** can for instance be a text or an image. The **padding**, **border** and **margin** each have a top, bottom, left and right part. 5-5-5

To calculate the total width of an element we can use the following formula:

```
margin-right + border-right + padding-right + width + padding-left +
border-left + margin-left
```

And so, to calculate the total height of an element the formula would be:

```
margin-top + border-top + padding-top + height + padding-bottom +
border-bottom + margin-bottom
```

One thing to keep in mind is that the `width` and `height` property cannot be set for inline element, only block elements. As we have seen before, a block element will start a new line and take all horizontal space. The default width for a block element is therefore 100%. But, since inline elements cannot have a fixed size, the `width` and `height` property does not apply to them. It is possible to set a specific height for a block element. If none is set, the element will grow and decrease vertically to accommodate its content.

CSS demo: Block vs inline

Understanding the difference between **block** content and **inline** content is absolutely central. These are controlled through the `display` property.

```
<div>Once upon a time there was a <strong>very</strong> scary gnome.</div>
```

```
strong {
    background-color: red;
    height: 50px;
    display: inline;
}
```

Variant 1/3:

Since `<strong>` is an inline element by default, it flows with the text. Thus, height has no meaning here, since it cannot have a fixed size.

Once upon a time, there was a very scary gnome.

Variant 2/3:

But change to block and see what happens!

Once upon a time, there was a
very
scary gnome.

Variant 3/3:

Now `inline-block`! Wow!

Once upon a time, there was a very scary gnome.

Depending on which display mode an element has, it will **behave differently** in a number of ways, and other CSS **properties get different meaning**.

CSS demo: Box sizing

Showing off different box sizing

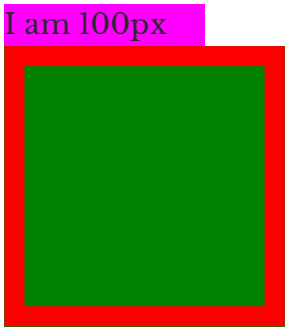
```
<div class="ruler">I am 100px</div>
<div class="box"></div>
```

```
.ruler {
    width: 100px;
    background-color: magenta;
}

.box {
    background-color: green;
    width: 100px;
    height: 100px;
    padding: 10px;
    border: 10px solid red;
    box-sizing: content-box;
}
```

Variant 1/2:

The default is called `content-box`, and doesn't include padding and border.

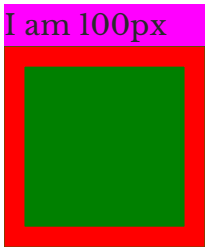
```
.ruler {
    width: 100px;
    background-color: magenta;
}

.box {
    background-color: green;
    width: 100px;
    height: 100px;
    padding: 10px;
    border: 10px solid red;
    box-sizing: content-box;
}
```



Variant 2/2:

If we *do* want to include padding and border, we use border-box.

```
.ruler {
    width: 100px;
    background-color: magenta;
}

.box {
    background-color: green;
    width: 100px;
    height: 100px;
    padding: 10px;
    border: 10px solid red;
    box-sizing: border-box;
}
```



CSS demo: Visibility

Showing off `visibility: none`

```
<div>Once upon a time there was a <strong>very</strong> scary gnome.</div>
```

```
strong {
    visibility: visible;
}
```

Variant 1/3:

All elements have `visible` as default visibility.

Once upon a time there was a **very** scary gnome.

Variant 2/3:

Note how it still takes up the same space!

```
strong {
    visibility: hidden;
}
```

Once upon a time there was a scary gnome.

Variant 3/3:

While if we set `display` to `none`, it takes up no space at all.

```
strong {
    display: none;
}
```

Once upon a time there was a scary gnome.

5-6. Property values

In previous examples, we have seen a few basic styles, like 5-6-1

```
body {
    background-color: red;
}
```

But **what values can they have**? Well, that depends.

Many properties have **unique values** that only they can have. 5-6-2

For example, the `color` property, which tells the browser what color the font in the selected element should have, may only be set to a color value.

```
body {
    color: red;
}
```

CSS colors can be specified using one of the following: 5-6-3

- Named color values
- Hexadecimal
- RGB
- RGA
- HSL
- HSLA

So, to make the font color in the body red, you can use one of the following:


```
body { color: red; }
body { color: #f00; }
body { color: #ff0000; }
body { color: rgb(255,0 ,0); }
body { color: rgb(100%, 0%, 0%); }
body { color: hsl(0, 100%, 50%); }

/* 50% translucent */
body { color: rgba(255, 0, 0, 0.5); }
body { color: hsla(0, 100%, 50%, 0.5); }
```

Let's look at an example of the `background-image` property!

CSS demo: Background image

Demonstrating background images

```
<div class="box"></div>
```

Variant 1/4:

The background is repeated by default.

```
.box {
    height: 200px;
    width: 600px;
    background-color: gold;
    background-image: url(resources/css3.png);
}
```

Variant 2/4:

We can tell it not to repeat the image.

```
.box {
    height: 200px;
    width: 600px;
    background-color: gold;
```

```
    background-image: url(resources/css3.png);
    background-repeat-x: no-repeat;
}
```

Variant 3/4:

The background can be positioned.

```
.box {
    height: 200px;
    width: 600px;
    background-color: gold;
    background-image: url(resources/css3.png);
    background-repeat-x: no-repeat;
    background-position: center;
}
```

Variant 4/4:

The image can be sized according to the box.

```
.box {
    height: 200px;
    width: 600px;
    background-color: gold;
    background-image: url(resources/css3.png);
```

```
    background-repeat-x: no-repeat;
    background-position: center;
    background-size: 100% 100%;
}
```

A lot of properties take a **length**, and these can have **different units**. *5-6-5*

These can be divided into **absolute** and **relative** units.

Absolute units include: *5-6-6*

- mm
- px
- pt

Relative units include: *5-6-7*

- em, relative to the parent font
- %

CSS demo: Length units

Testing different length units.

```
<div class="box">Testing</div>
```

Variant 1/4:

Pixels, px, is perhaps the most common unit

```
.box {
    width: 70px;
    background-color: magenta;
}
```

Testing!

Variant 2/4:

A point, pt, is 1/72 inches (1 inch=2.54 cm)

```
.box {
    width: 70pt;
    background-color: magenta;
}
```

Testing!

Variant 3/4:

We can also use regular units such as millimeter, mm.

```
.box {
    width: 70mm;
    background-color: magenta;
}
```

Testing!

Variant 4/4:

Percentage in this case means percent of the parent width.

```
.box {
    width: 70%;
```

```
    background-color: magenta;
}
```

Testing!

CSS demo: The percent unit

Showing off the % unit

```
<div class="shell">Container
    <div class="box">Box!</div>
</div>
```

Variant 1/2:

The percentage relates to the parent width.

```
.shell {
    width: 200px;
    background-color: magenta;
}

.box {
    width: 50%;
    background-color: red;
}
```

Container
Box!

Variant 2/2:

Change the parent and the box changes to!

```
.shell {
    width: 500px;
    background-color: magenta;
}

.box {
    width: 50%;
    background-color: red;
}
```

Container
Box!

CSS demo: The EM unit

Showing off the `em` unit

```
<div class="shell">Testing ems
    <div class="box"></div>
</div>
```

Variant 1/2:

The `em` is relative to the parent font.

```
.shell {
    font-size: 14px;
}

.box {
    width: 12em;
    height: 3em;
    background-color: red;
}
```

Testing ems

Variant 2/2:

We increase the parent font, and the box grows!

```
.shell {
    font-size: 20px;
}

.box {
    width: 12em;
    height: 3em;
    background-color: red;
}
```

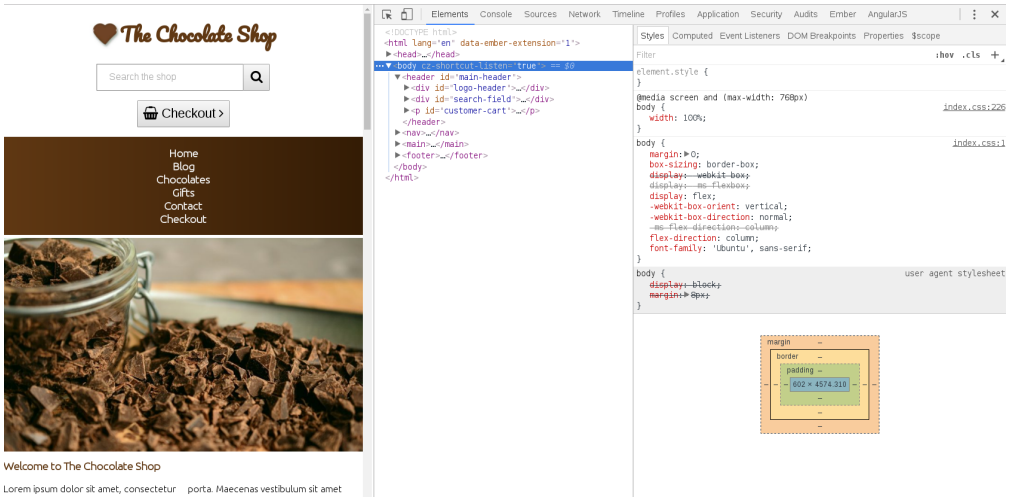
Testing ems

Learning all the details is impossible. You **must have a reference** such as MDN! *5-6-8*

5-7. The browser developer tools

Today we have **very powerful developer tools** built into almost all browsers. We'll be looking at Chrome, but there are analogs in the others. *5-7-1*

They **show you the element tree** and **provide information on specific elements**. *5-7-2*



This is especially useful for *5-7-3*

- **CSS debugging**, as you can see exactly **what styles are applied** and from where and with **which selectors** they came.
- **CSS experimenting**, as you can toggle the styles, try out values and even add completely new styles.

5-8. Pseudo-elements

You've already met pseudo-classes. However, the **pseudo-elements** do something we haven't seen yet - they **add new content** to the document! *5-8-1*

Say we have this <p> element: *5-8-2*

```
<p>Eeeexiiit <em>light!</em> Eeenteeer <em>niiight!</em></p>
```

And we target it with this CSS:

```
p:before {
    content: "♬ ";
}

p:after {
    content: " ♫♩";
}
```

Then we get this output: 5-8-3

Eeeexiiit *light!* Eeenteeer *niiight!*

Note however that the notes **don't actually come before and after** the <p> element, but rather **become the youngest and oldest children** 5-8-4

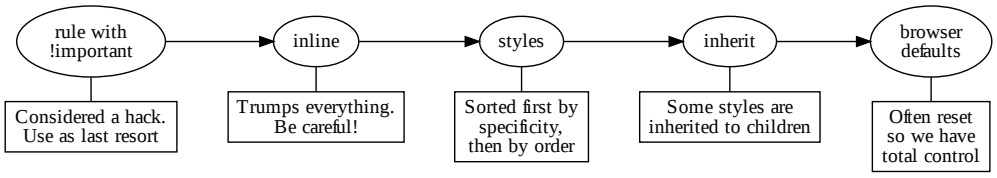
In other words, `:before` and `:after` are **misleading names**, they should have been called something else. Also, there should be an *actual* before and after, but there isn't 5-8-5

But still, the pseudo-elements are a powerful tool when the moment is right. They can be used to style a specific part of an element or, like in the example above, to insert new content. 5-8-6

5-9. Order

Q So, styles can be defined in many different places. But, **what happens when the same style is applied in different ways**? 5-9-1

A The browser follows a strict pecking order: 5-9-2



There were some things in there that we don't yet know about. First, `!important` - this is a keyword you can **add to the end of a rule** to bring it to the top of the pecking order. 5-9-3

```
.special {
    background-color: yellow !important;
}
```

But note that this is **considered bad practice**, just as **inline styles are considered harmful**. 5-9-4

So if we **avoid those two** the primary order is **specificity**. But, what is that? 5-9-5

Specificity is the "**weight" of the rule**, which is calculated according to a formula. The gist: 5-9-6

1. **Ids** are the heaviest
2. then **classes, attributes and pseudo-classes**
3. and finally **elements and pseudo-elements**

In short, the more **specific** a rule, the higher the **specificity**. If two rules are equal, the latter wins. 5-9-7

Say you have this style... 5-9-8

```
p:first-line {
    margin-left: 2em;
}
```

...but you are also using an **external CSS sheet** which **overrides your style**.

Then you need to **make your style more specific**, often by **prefixing it with ancestor specifications** that you otherwise don't need: 5-9-9

```
body div#app-wrapper p:first-line {
    margin-left: 2em;
}
```

A final note; since **browser defaults can vary**, it is common to use a **reset sheet** to be (more) sure that everything looks the same in different browsers. 5-9-10

An **old but popular sheet** is the one by CSS guru Eric Meyer, which you can find here: https://meyerweb.com/eric/tools/css/reset/:

5-10. Good behaviour

We've touched upon most of these points already, but let's **gather up the stone tablets of CSS behaviour rules**! 5-10-1

Thou shalt not use visual elements 5-10-2

- `<br>` for a linebreak
- `<hr>` for a horisontal ruler

These are **FORBIDDEN**!

Thou shalt not use visual attributes 5-10-3

- `<div bgcolor="red">`
- `<div height="300">`

These are **VERBOTEN**!

Thou shalt keep the `html` as clean as possible 5-10-4

Instead of doing this...

```
<p class="firstParagraph">Little red riding hood...</p>
<p>Then came the big bad wolf...</p>
```

...and this...

```
.firstParagraph {
  background-color: #CCC;
}
```

...you should **skip the class** and just do this: 5-10-5

```
p:first-child {
  background-color: #CCC;
}
```

Try to **solve the selection without cluttering the html**, as much as possible.

Thou shalt keep your selectors simple 5-10-6

Don't make your selectors more specific than they need to be!

Q **Question**: Ehm, doesn't that commandment collide with the previous one? Isn't it **easier to avoid cluttering the html if we have more complex selectors**? And **easier to write short selectors if we add some classes or extra divs to the html**? 5-10-7

A **Answer**: Yes. :D 5-10-8

Thou shalt keep your CSS organized 5-10-9

Keeping your CSS organized is hard!

Still, **you must try**! Use **comments**, **file separation** and **structure**. A good tip is to **do whatever Nicole Sullivan says**: http://www.stubbornella.org/

And now for the **most important commandment of all**. 5-10-10

Are you ready? 5-10-11

Thou must be patient! 5-10-12

You now **know the basic rules**, but probably feel a **whole lot of frustration**.

This is because **I have taught you the grammar**, but you still have to **learn the vocabulary**. To this, alas, there are **no shortcuts**.