CSS syntax

Previous Overview: Introduction to CSS Next

Next up, we dive into CSS syntax in a lot more detail, looking at how *properties* and their *values* form into *declarations*, multiple declarations form into *declaration blocks*, and declaration blocks and *selectors* form into complete *CSS rules*. We round off the article by looking at other CSS syntax features such as comments and whitespace.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of How CSS works.
Objective:	To learn CSS's fundamental syntax structures in detail.

Note: CSS is a declarative language, which makes its syntax fairly easy and straightforward to understand. In addition, it also has a very nice error recovery system that allows you to make mistakes without breaking everything — for example declarations that aren't understood are generally just ignored. The downside is that it can be harder to understand where errors are coming from. Read on, and all will become clear eventually.

A touch of vocabulary

At its most basic level, CSS consists of two building blocks:

- **Properties**: Human-readable identifiers that indicate which stylistic features (e.g. font, width, background color) you want to change.
- **Values**: Each specified property is given a value, which indicates how you want to change those stylistic features (e.g. what you want to change the font, width or background color to.)

A property paired with a value is called a *CSS declaration*. CSS declarations are put within *CSS Declaration Blocks*. And finally, CSS declaration blocks are paired with *selectors* to produce *CSS Rulesets* (or *CSS Rules*).

Before getting too deep in theory and written explanation, let's look at a concrete example (we saw something very similar in our previous article.)

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>

    <ul>
      <li>This is</li>
      <li>a list</li>
    </ul>
  </body>
</html>
```

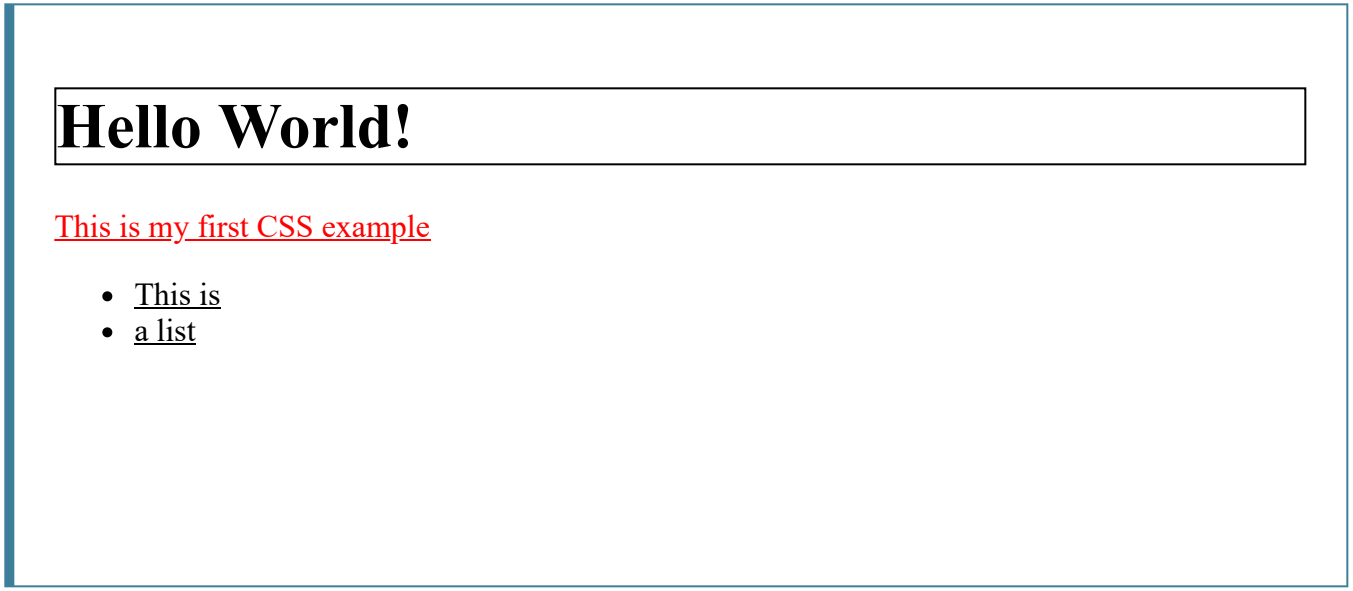
And the CSS file:

```css
h1 {
  colour: blue;
  background-color: yellow;
  border: 1px solid black;
}

p {
  color: red;
}
```

```
10
11  p, li {
12    text-decoration: underline;
13  }
```

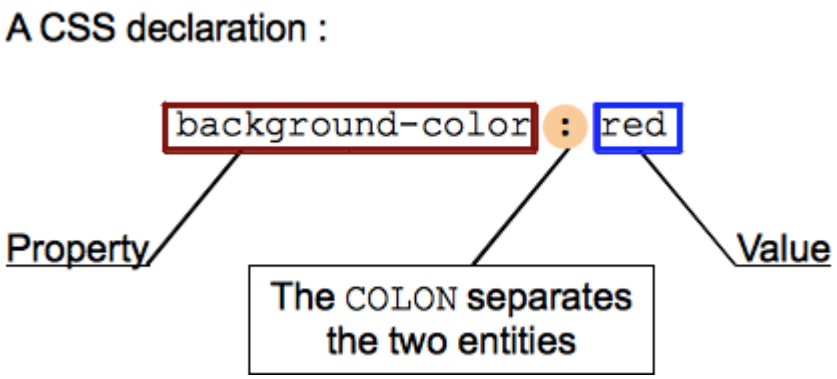
Combining these two gives us the following result:



Let's look at the syntax in more detail.

CSS declarations

Setting CSS properties to specific values is the core function of the CSS language. The CSS engine calculates which declarations apply to every single element of a page in order to appropriately lay it out and style it. What is important to remember is that both properties and values are case-sensitive in CSS. The property and value in each pair is separated by a colon (`:`).



There are more than 300 different properties in CSS and nearly an infinite number of different values. Not all pairs of properties and values are allowed; each property has a specific list of valid values defined for it.

> **Important**: If a property is unknown or if a value is not valid for a given property, the declaration is deemed *invalid* and is wholly ignored by the browser's CSS engine.

> **Important**: In CSS (and other web standards), US spelling has been agreed on as the standard to stick to where uncertainty arises. For example, `color` should *always* be spelled `color`. `colour` won't work.

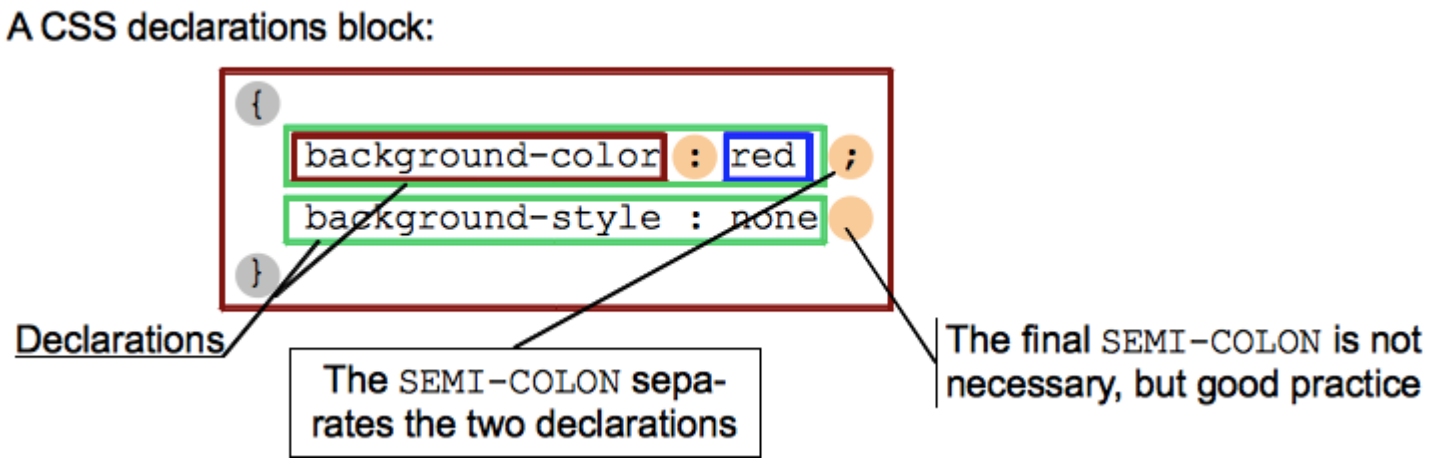
Active learning: Spot the declarations

In the example above, there are **five** separate CSS declarations. Can you identify the invalid (incorrect or wrong) declaration and work out why it's invalid?

CSS declaration blocks

Declarations are grouped in **blocks**, with each set of declarations being wrapped by an opening curly brace, (`{`) and a closing one (`}`).

Each declaration contained inside a **declaration block** has to be separated by a semi-colon (`;`), otherwise the code won't work (or will at least give unexpected results.) The last declaration of a block doesn't need to be terminated by a semi-colon, though it is often considered *good style* to do so as it prevents forgetting to add it when extending the block with another declaration.



> **Note**: Blocks can sometimes be nested; in such cases opening and closing braces must be nested logically, in the same fashion as the tags of nested HTML elements. The most common example you'll come across are *@-rules,* which are blocks beginning with an @ identifier like `@media`, `@font-face`, etc (See *CSS Statements* below).

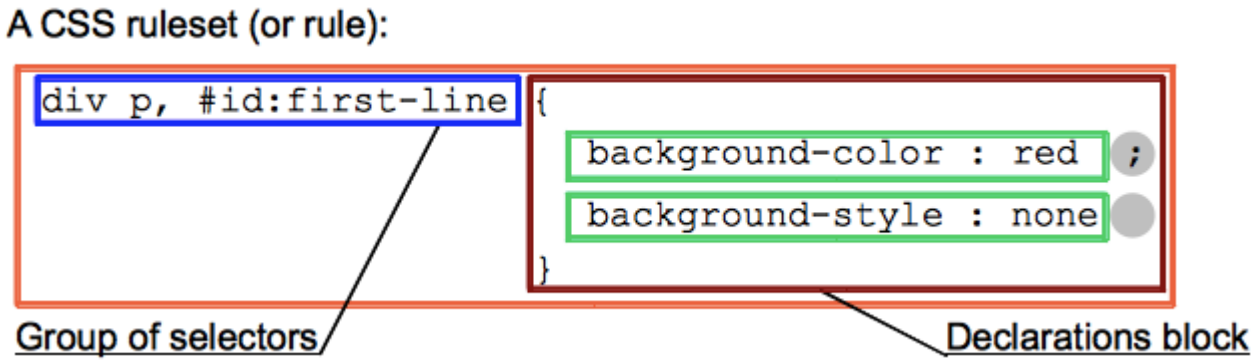
> **Note**: A declaration block may be empty — this is perfectly valid.

Active learning: Where are the declaration blocks?

In the above example, have you been able to identify the **three** separate CSS declaration blocks?

CSS selectors and rules

We are missing one part of the puzzle — we need to discuss how to tell our declaration blocks which elements they should be applied to. This is done by prefixing each declaration block with a **selector** — a pattern that matches some elements on the page. The associated declarations will be applied to those elements only. The selector plus the declaration block is called a **ruleset**, or often simply just a **rule**.



Selectors can get very complicated — you can make a rule match multiple elements by including multiple selectors separated by commas (a group,) and selectors can be chained together, for example *I want to select any element with a class of "blah", but only if it is inside an <article> element, and only while it is being hovered by the mouse pointer*. Don't worry — things will become clearer as you become more experienced with CSS, and we'll explain selectors in great detail in the next article, Selectors.

An element may be matched by several selectors, therefore several rules may set a given property multiple times. CSS defines which one has precedence over the others and must be applied: this is called the **cascade algorithm**, and you'll learn more about how it works in Cascade and inheritance.

> **Important**: If a single basic selector in a chain or group is invalid, like when using an unknown pseudo-element or pseudo-class, the *group of selectors* is still valid, except for the invalid selector which will be ignored.

Active learning: Spot the group of selectors

In our example, have you been able to identify the rule that will apply to **two** different types of elements?

CSS statements

CSS Rules are the main building blocks of a style sheet — the most common block you'll see in CSS. But there are other types of block that you'll come across occasionally — CSS rules are one type of so-called CSS statements. The other types are as follows:

- **At-rules** are used in CSS to convey metadata, conditional information, or other descriptive information. They start with an at sign (`@`), followed by an identifier to say what kind of rule it is, then a syntax block of some kind, ending with a semi-colon (;). Each type of at-rule, defined by the identifier, will have its own internal syntax and semantics. Examples include:
  - `@charset` and `@import` (metadata)
  - `@media` or `@document` (conditional information, also called nested statements, see below.)
  - `@font-face` (descriptive information)

  Specific syntax example:

  ```
  1 @import 'custom.css';
  ```

This at-rule imports another CSS file into the current CSS.

- **Nested statements** are a specific subset of at-rule, the syntax of which is a nested block of CSS rules that will only be applied to the document if a specific condition is matched:
  - The `@media` at-rule content is applied only if the device which runs the browser matches the expressed condition;
  - the `@supports` at-rule content is applied only if the browser actually supports the tested feature;
  - the `@document` at-rule content is applied only if the current page matches some conditions.

  Specific syntax example:

```
@media (min-width: 801px) {
  body {
    margin: 0 auto;
    width: 800px;
  }
}
```

  The above nested statement only applies the nested rule when the page's width exceeds 800 pixels.

You'll learn more about some types of at-rules/nested statements at appropriate places in the course.

> **Important**: Any statement which isn't a ruleset, an at-rule, or a nested statement is invalid and therefore ignored.

Beyond syntax: make CSS readable

As you can see, CSS syntax is not that hard to write: you write some rules and if you write them wrong, they will just be ignored. However, even if that works, there are some best practices worth knowing to make your CSS code easier to use and maintain.

White space

White space means actual spaces, tabs and new lines. You can add white space to make your stylesheets more readable.

In the same manner as HTML, the browser tends to ignore much of the whitespace inside your CSS; a lot of the whitespace is just there to aid readability. In our first example below we have each declaration (and rule start/end) on its own line — this is arguably a good way to write CSS, as it makes it easy to maintain and understand:

```
1  body {
2    font: 1em/150% Helvetica, Arial, sans-serif;
3    padding: 1em;
4    margin: 0 auto;
5    max-width: 33em;
6  }
7
8  @media (min-width: 70em) {
9    body {
10     font-size: 130%;
11   }
12 }
13
```

```
h1 {
  font-size: 1.5em;
}

div p, #id:first-line {
  background-color: red;
  background-style: none
}

div p {
  margin: 0;
  padding: 1em;
}

div p + p {
  padding-top: 0;
}
```

You could write exactly the same CSS like so, with most of the whitespace removed — this is functionally identical to the first example, but I'm sure you'll agree that it is somewhat harder to read:

```
body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin: 0 auto; max-width: 33em;}
@media (min-width: 70em) { body {font-size: 130%;} }

h1 {font-size: 1.5em;}

div p, #id:first-line {background-color: red; background-style: none}

```

```
8 | div p {margin: 0; padding: 1em;}
  | div p + p {padding-top: 0;}
```

The code layout you choose is usually a personal preference, although when you start to work in teams, you may find that the existing team has its own style guide that specifies an agreed convention to follow.

The whitespace you do need to be careful of in CSS is the whitespace around the properties and their values. For example, the following is valid CSS:

```
1 | margin: 0 auto;
2 | padding-left: 10px;
```

But the following is invalid:

```
1 | margin: 0auto;
2 | padding- left: 10px;
```

Because `0auto` is not recognised as a valid value for the margin property (`0` and `auto` are two separate values,) and the browser does not recognise `padding-` as a valid property. So you should always make sure to separate distinct values from one another by at least a space, but keep property names/values together as one unbroken string.

Comments

As with HTML, you are encouraged to make comments in your CSS, to help you understand how your code works when coming back to it after several months, and to help others

understand it. Comments are also useful for temporarily *commenting out* certain parts of the code for testing purposes, for example if you are trying to find which part of your code is causing an error.

Comments in CSS begin with `/*` and end with `*/`.

```
/* Handle basic element styling */
/* ------------------------------------------------------------------------------------- */
body {font: 1em/150% Helvetica, Arial, sans-serif; padding: 1em; margin: 0 auto; max-width: 33em;}
@media (min-width: 70em) {
  /* Let's special case the global font size. On large screen or window,
     we increase the font size for better readability */
  body {font-size: 130%;}
}

h1 {font-size: 1.5em;}

/* Handle specific elements nested in the DOM  */
/* ------------------------------------------------------------------------------------- */
div p, #id:first-line {background-color: red; background-style: none}
div p                 {margin           :   0; padding          : 1em;}
div p + p             {padding-top      :   0;                       }
```

Shorthand

Some properties like `font`, `background`, `padding`, `border`, and `margin` are called **shorthand properties** — this is because they allow you to set several property values in a single line, saving time and making your code neater in the process.

For example, this line:

```
/* in shorthand like padding and margin, the values are applied
   in the order top, right, bottom, left (the same order as an analog clock). There are also other
   shorthand types, for example two values, which set for example
   the padding for top/bottom, then left/right */
padding: 10px 15px 15px 5px;
```

Does the same thing as all these:

```
padding-top: 10px;
padding-right: 15px;
padding-bottom: 15px;
padding-left: 5px;
```

Whereas this line:

```
background: red url(bg-graphic.png) 10px 10px repeat-x fixed;
```

Does the same thing as all these:

```
background-color: red;
background-image: url(bg-graphic.png);
background-position: 10px 10px;
background-repeat: repeat-x;
background-scroll: fixed;
```

We won't attempt to teach these exhaustively — you'll come across many examples later on in the course, and you are advised to look up the shorthand property names in our CSS reference to find out more.

What's next

At this point, you should now understand the fundamentals of CSS syntax necessary to write a working style sheet that's easy to maintain over time. In the next article we will dig deep into CSS selectors, looking at the different ones available and how they work.

Previous | Overview: Introduction to CSS | Next

In this module

- Combinators and multiple selectors
- CSS values and units
- Cascade and inheritance
- The box model
- Debugging CSS
- Fundamental CSS comprehension