Fundamentals of CSS

Darkhan Zhursin
darkhan.zhursin@jihc.kz

Introduction: Fundamentals of CSS

After this unit, you will be able to:

- Understand how CSS is used for web development
- Use CSS to add initial styling to your website
- Understand the Box Model in CSS
- Add positioning using CSS
- Read CSS documentation

Intro to CSS

While HTML is the fundamental structure of every web page, it can be visually unappealing on its own. Cascading Style Sheets or *CSS* is a language web developers use to style the HTML content on a web page. If you’re interested in modifying colors, font types, font sizes, images, element positioning, and more, CSS is the tool for the job!

In this lesson, you’ll learn how to set up your CSS file structure and select which HTML elements you wish to style.

CSS Anatomy:

CSS ruleset	CSS inline style

selector
`p {`
`    color: blue;`
`}`
	opening tag
	`<p style='color: blue;'>Hello World!</p>`

declaration block
`p {`
`    color: blue;`
`}`
	attribute
	`<p style='color: blue;'>Hello World!</p>`

declaration
`p {`
`    color: blue;`
`}`
	declaration
	`<p style='color: blue;'>Hello World!</p>`

property
`p {`
`    color: blue;`
`}`
	property
	`<p style='color: blue;'>Hello World!</p>`

value
`p {`
`    color: blue;`
`}`
	value
	`<p style='color: blue;'>Hello World!</p>`

Inline Styles

Although CSS is a different language than HTML, it’s possible to write CSS code directly within HTML code using *inline styles*.

To style an HTML element, you can add the `style` attribute directly to the opening tag. After you add the attribute, you can set it equal to the CSS style(s) you’d like applied to that element.

```
<p style='color: red;'>I'm learning to code!</p>
```

The code in the example above demonstrates how to use inline styling. The paragraph element has a `style` attribute within its opening tag. Next, the `style` attribute is set equal to `color: red;`, which will set the color of the paragraph text to red within the browser.

If you’d like to add *more* than one style with inline styles, simply keep adding to the `style` attribute. Make sure to end the styles with a semicolon (`;`).

```
<p style='color: red; font-size: 20px;'>I'm learning to code!</p>
```

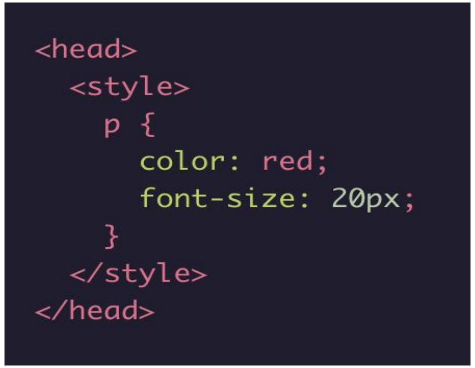
It’s important to know that inline styles are a quick way of directly styling an HTML element, but are rarely used when creating websites. But you may encounter circumstances where inline styling is necessary, so understanding how it works, and recognizing it in HTML code is good knowledge to have. Soon you’ll learn the proper way to add CSS code!

Internal Stylesheet

As previously stated, inline styles are not the best way to style HTML elements. If you wanted to style, for example, multiple `<h1>` elements, you would have to add inline styling to each element manually. In addition, you would also have to maintain the HTML code when additional `<h1>` elements are added.

Fortunately, HTML allows you to write CSS code in its own dedicated section with a `<style>` element nested inside of the `<head>` element. The CSS code inside the `<style>` element is often referred to as an *internal stylesheet*.

An internal stylesheet has certain benefits and use cases over inlines styles, but once again, it's not best practice. Understanding how to use internal stylesheets is nonetheless helpful knowledge to have. To create an internal stylesheet, a `<style>` element must be placed inside of the `<head>` element:



```
<head>
  <style>
    p {
      color: red;
      font-size: 20px;
    }
  </style>
</head>
```

External Stylesheet

Developers avoid mixing code by storing HTML and CSS code in separate files (HTML files contain only HTML code, and CSS files contain only CSS code).

You can create an external stylesheet by using the `.css` file name extension, like so: `style.css`

With an external stylesheet, you can write all the CSS code needed to style a page without sacrificing the readability and maintainability of your HTML file.

Linking the CSS File

Perfect! We successfully separated structure (HTML) from styling (CSS), but the web page still looks bland. Why?

When HTML and CSS codes are in separate files, the files must be linked. Otherwise, the HTML file won't be able to locate the CSS code, and the styling will not be applied.

You can use the `<link>` element to link HTML and CSS files together. The `<link>` element must be placed within the head of the HTML file. It is a self-closing tag and requires the following attributes:

1. `href` — like the anchor element, the value of this attribute must be the address, or path, to the CSS file.
2. `rel` — this attribute describes the relationship between the HTML file and the CSS file. Because you are linking to a stylesheet, the value should be set to `stylesheet`.

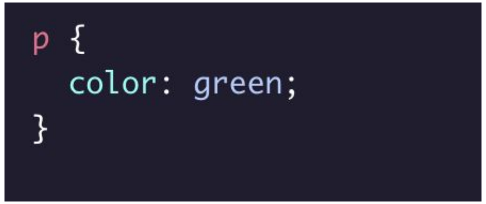
When linking an HTML file and a CSS file together, the `<link>` element will look like the following:

```
<link href='./style.css' rel='stylesheet'>
```

SELECTORS: Type

Remember that *declarations* are a fundamental part of CSS because they apply a style to a selected element. But how do you decide which elements will get the style? With a *selector*.

A selector is used to target the specific HTML element(s) to be styled by the declaration. One selector you may already be familiar with is the *type* selector. Just like its name suggests, the type selector matches the *type* of the element in the HTML document.



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

This is an instance of using the type selector! The element type is `p`, which comes from the HTML `<p>` element.

Some important notes on the type selector:

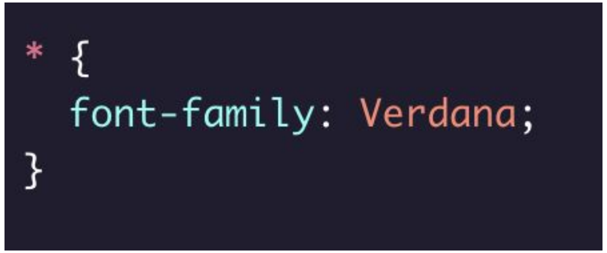
- The type selector does not include the angle brackets.
- Since element types are often referred to by their opening tag name, the type selector is sometimes referred to as the *tag name* or *element* selector.

Universal

You learned how the *type selector* selects all elements of a given type. Well, the *universal selector* selects all elements of *any* type.

Targeting all of the elements on the page has a few specific use cases, such as resetting default browser styling, or selecting all children of a parent element. Don't worry if you don't understand the use cases right now—we will get to them later on in our Learn CSS journey.

The universal selector uses the `*` character in the same place where you specified the type selector in a ruleset, like so:



```css
* {
  font-family: Verdana;
}
```

In the code, every text element on the page will have its font changed to `Verdana`.

Class

CSS is not limited to selecting elements by their type. As you know, HTML elements can also have attributes. When working with HTML and CSS a *class* attribute is one of the most common ways to select an element.

For example, consider the following HTML: `<p class='brand'>Sole Shoe Company</p>`

The paragraph element in the example above has a `class` attribute within the opening tag of the`<p>` element. The `class` attribute is set to `'brand'`. To select this element using CSS, we can create a ruleset with a class selector of `.brand .`

```
.brand {

}
```

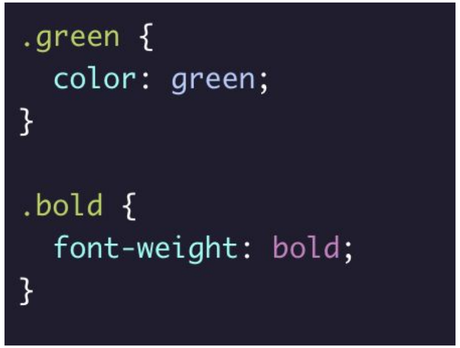
To select an HTML element by its class using CSS, a period (`.`) must be prepended to the class's name. In the example above, the class is `brand`, so the CSS selector for it is `.brand`.

Multiple Classes

We can use CSS to select an HTML element’s `class` attribute by name. And so far, we’ve selected elements using only one class name per element. If every HTML element had a single class, all the style information for each element would require a new class.

Luckily, it’s possible to add more than one class name to an HTML element’s `class` attribute.

For instance, perhaps there’s a heading element that needs to be green and bold. You could write two CSS rulesets like so:



```css
.green {
  color: green;
}

.bold {
  font-weight: bold;
}
```

Then, you could include both of these classes on one HTML element like this: `<h1 class='green bold'> ... </h1>`

We can add multiple classes to an HTML element’s `class` attribute by separating them with a space. This enables us to mix and match CSS classes to create many unique styles without writing a custom class for every style combination needed.

ID

Oftentimes it’s important to select a single element with CSS to give it its own unique style. If an HTML element needs to be styled uniquely, we can give it an ID using the `id` attribute: `<h1 id='large-title'> ... </h1>`

In contrast to `class` which accepts multiple values, and can be used broadly throughout an HTML document, an element’s `id` can only have a single value, and only be used once per page.

To select an element’s ID with CSS, we prepend the `id` name with a number sign (`#`). For instance, if we wanted to select the HTML element in the example above, it would look like this: `#large-title {}`

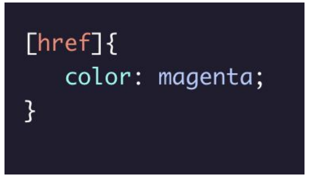
The `id` name is `large-title`, therefore the CSS selector for it is `#large-title`.

Attribute

You may remember that some HTML elements use attributes to add extra detail or functionality to the element. Some familiar attributes may be `href` and `src`, but there are many more—including `class` and `id`!

The *attribute selector* can be used to target HTML elements that already contain attributes. Elements of the same type can be targeted differently by their attribute or attribute value. This alleviates the need to add new code, like the `class` or `id` attributes.

Attributes can be selected similarly to types, classes, and IDs.



```
[href]{
   color: magenta;
}
```

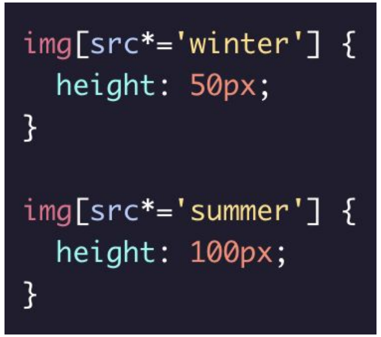
The most basic syntax is an attribute surrounded by square brackets. In the above example: `[href]` would target all elements with an `href` attribute and set the `color` to `magenta`.

And it can get more granular from there by adding type and/or attribute values. One way is by using `type[attribute*=value]`. In short, this code selects an element where the attribute contains any instance of the specified value. Let’s take a look at an example.22

```html
<img src='/images/seasons/cold/winter.jpg'>
```

```html
<img src='/images/seasons/warm/summer.jpg'>
```

The HTML code above renders two `<img>` elements, each containing a `src` attribute with a value equaling a link to an image file.



```css
img[src*='winter'] {
  height: 50px;
}

img[src*='summer'] {
  height: 100px;
}
```

Now take a look at the above CSS code. The *attribute selector* is used to target each image individually.

- The first ruleset looks for an `img` element with an attribute of `src` that contains the string `'winter'`, and sets the `height` to `50px`.
- The second ruleset looks for an `img` element with an attribute of `src` that contains the string `'summer'`, and sets the `height` to `100px`.

Notice how no new HTML markup (like a class or id) needed to be added, and we were still able to modify the styles of each image independently. This is one advantage to using the attribute selector!

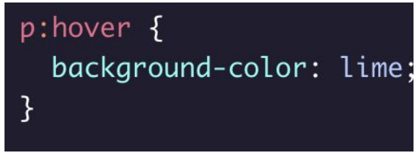
Pseudo-class

You may have observed how the appearance of certain elements can change, or be in a different state, after certain user interactions. For instance:

- When you click on an `<input>` element, and a blue border is added showing that it is in *focus*.
- When you click on a blue `<a>` link to *visit* to another page, but when you return the link's text is purple.
- When you're filling out a form and the submit button is grayed out and *disabled*. But when all of the fields have been filled out, the button has color showing that it's *active*.

These are all examples of pseudo-class selectors in action! In fact, `:focus`, `:visited`, `:disabled`, and `:active` are all pseudo-classes. Factors such as user interaction, site navigation, and position in the document tree can all give elements a different state with pseudo-class.

A pseudo-class can be attached to any selector. It is always written as a colon `:` followed by a name. For example `p:hover`.



```
p:hover {
  background-color: lime;
}
```

In the code, whenever the mouse hovers over a paragraph element, that paragraph will have a lime-colored background.

Classes and IDs

CSS can select HTML elements by their type, class, and ID. CSS classes and IDs have different purposes, which can affect which one you use to style HTML elements.

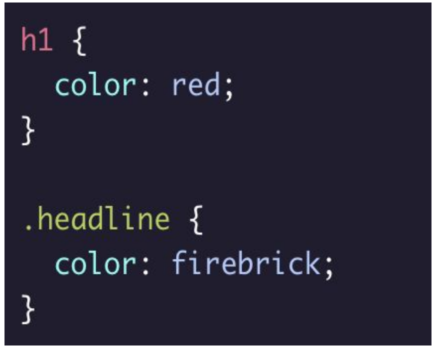
CSS classes are meant to be reused over many elements. By writing CSS classes, you can style elements in a variety of ways by mixing classes. For instance, imagine a page with two headlines. One headline needs to be bold and blue, and the other needs to be bold and green. Instead of writing separate CSS rules for each headline that repeat each other's code, it's better to write a `.bold` CSS rule, a `.green` CSS rule, and a `.blue` CSS rule. Then you can give one headline the `bold green` classes, and the other the `bold blue` classes.

While classes are meant to be used many times, an ID is meant to style only one element. IDs override the styles of types and classes. Since IDs override these styles, they should be used sparingly and only on elements that need to always appear the same.

Specificity

Specificity is the order by which the browser decides which CSS styles will be displayed. A best practice in CSS is to style elements while using the lowest degree of specificity so that if an element needs a new style, it is easy to override.

IDs are the most specific selector in CSS, followed by classes, and finally, type. For example, consider the following HTML and CSS: `<h1 class='headline'>Breaking News</h1>`



```
h1 {
  color: red;
}

.headline {
  color: firebrick;
}
```

In the example code, the color of the heading would be set to `firebrick`, as the class selector is more specific than the type selector. If an ID attribute (and selector) were added to the code above, the styles within the ID selector's body would override all other styles for the heading.

Over time, as files grow with code, many elements may have IDs, which can make CSS difficult to edit since a new, more specific style must be created to change the style of an element.

To make styles easy to edit, it's best to style with a type selector, if possible. If not, add a class selector. If that is not specific enough, then consider using an ID selector.

Chaining

When writing CSS rules, it’s possible to require an HTML element to have two or more CSS selectors at the same time.

This is done by combining multiple selectors, which we will refer to as chaining. For instance, if there was a `special` class for `<h1>` elements, the CSS would look like below: **h1.special { }**

The code above would select only the `<h1>` elements with a class of `special`. If a `<p>` element also had a class of `special`, the rule in the example would not style the paragraph.

Descendant Combinator

In addition to chaining selectors to select elements, CSS also supports selecting elements that are nested within other HTML elements, also known as *descendants*. For instance, consider the following HTML:

```
<ul class='main-list'>

  <li> ... </li>

  <li> ... </li>

  <li> ... </li>

</ul>
```

The nested `<li>` elements are descendants of the `<ul>` element and can be selected with the *descendant combinator* like so:

```
.main-list li { }
```

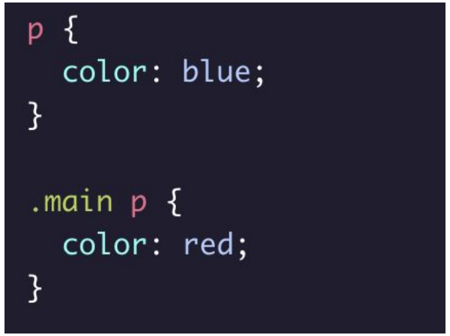
In the example above, `.main-list` selects the element with the `.main-list` class (the `<ul>` element). The descendant `<li>`'s are selected by adding `li` to the selector, separated by a space. This results in `.main-list li` as the final selector.

Selecting elements in this way can make our selectors even more specific by making sure they appear in the context we expect.

Chaining and Specificity

Adding more than one tag, class, or ID to a CSS selector increases the specificity of the CSS selector.

For instance, consider the following CSS:



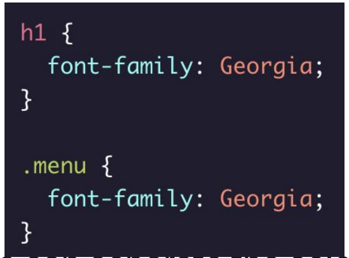
```css
p {
  color: blue;
}

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

Both of these CSS rules define what a `<p>` element should look like. Since `.main p` has a class and a `p` type as its selector, only the `<p>` elements inside the `.main` element will appear `red`. This occurs despite there being another more general rule that states `<p>` elements should be `blue`.

Multiple Selectors

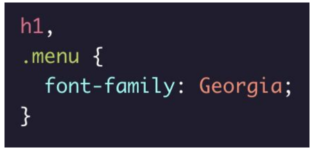
In order to make CSS more concise, it’s possible to add CSS styles to multiple CSS selectors all at once. This prevents writing repetitive code.

For instance, the following code has repetitive style attributes:



```css
h1 {
  font-family: Georgia;
}

.menu {
  font-family: Georgia;
}
```

Instead of writing `font-family: Georgia` twice for two selectors, we can separate the selectors by a comma to apply the same style to both, like this:



```css
h1,
.menu {
  font-family: Georgia;
}
```

By separating the CSS selectors with a comma, both the `<h1>` elements and the elements with the `menu` class will receive the `font-family: Georgia` styling.

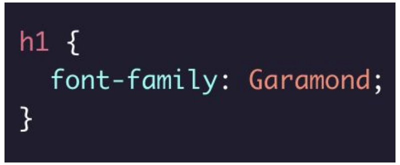
Visual Rules

The purpose of CSS is to add style to web page, and each element on the page can have many style properties. Some of the basic properties relate to the size, style, and color of the element. In this lesson, you’ll learn some fundamental CSS visual rules that you can use to start styling web page elements!

Font Family

If you’ve ever used word processing software, like Microsoft Word or Google Docs, chances are that you probably also used a feature that allowed you to change the font you were typing in. Font refers to the technical term typeface, or *font family*.

To change the typeface of text on your web page, you can use the `font-family` property.

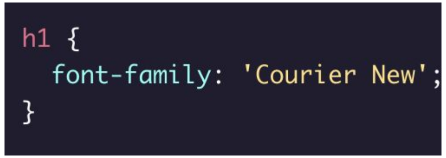


```
h1 {
  font-family: Garamond;
}
```

In the example above, the font family for all main heading elements has been set to `Garamond`.

When setting typefaces on a web page, keep the following points in mind:

- The font specified must be installed on the user’s computer or downloaded with the site.
- Web safe fonts are a group of fonts supported across most browsers and operating systems.
- Unless you are using web safe fonts, the font you choose may not appear the same between all browsers and operating systems.
- When the name of a typeface consists of more than one word, it’s a best practice to enclose the typeface’s name in quotes, like so:

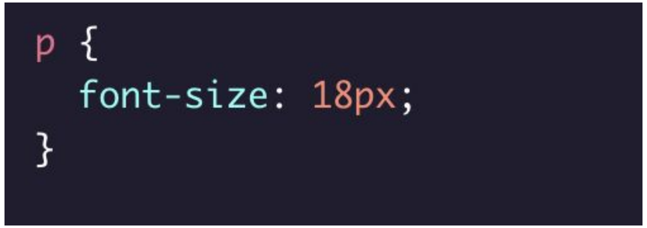


```
h1 {
  font-family: 'Courier New';
}
```

Font Size

Changing the typeface isn't the only way to customize the text. Oftentimes, different sections of a web page are highlighted by modifying the *font size*.

To change the size of text on your web page, you can use the `font-size` property.

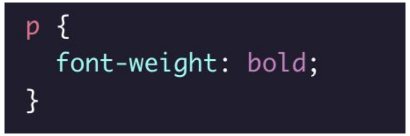


```css
p {
  font-size: 18px;
}
```

In the example above, the `font-size` of all paragraphs was set to `18px`. `px` means pixels, which is one way to measure font size.

Font Weight

In CSS, the `font-weight` property controls how bold or thin text appears.



```
p {
  font-weight: bold;
}
```

In the example above, all paragraphs on the web page would appear bolded.

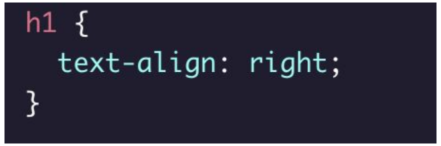
The `font-weight` property has another value: `normal`. Why does it exist?

If we wanted *all* text on a web page to appear bolded, we could select all text elements and change their font weight to `bold`. If a certain section of text was required to appear normal, however, we could set the font weight of that particular element to `normal`, essentially shutting off bold for that element.

Text Align

No matter how much styling is applied to text (typeface, size, weight, etc.), the text always appears on the left side of the container in which it resides.

To align text we can use the `text-align` property. The `text-align` property will align text to the element that holds it, otherwise known as its *parent*.



```css
h1 {
  text-align: right;
}
```

The `text-align` property can be set to one of the following commonly used values:

- `left` — aligns text to the left side of its parent element, which in this case is the browser.
- `center` — centers text inside of its parent element.
- `right` — aligns text to the right side of its parent element.
- `justify`— spaces out text in order to align with the right and left side of the parent element.

Color and Background Color

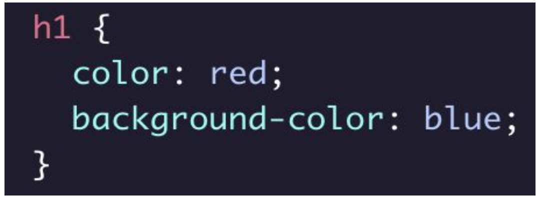
Before discussing the specifics of color, it’s important to make two distinctions about color. Color can affect the following design aspects:

- Foreground color
- Background color

Foreground color is the color that an element appears in. For example, when a heading is styled to appear green, the *foreground color* of the heading has been styled. Conversely, when a heading is styled so that its background appears yellow, the *background color* of the heading has been styled.

In CSS, these two design aspects can be styled with the following two properties:

- `color`: this property styles an element’s foreground color
- `background-color`: this property styles an element’s background color



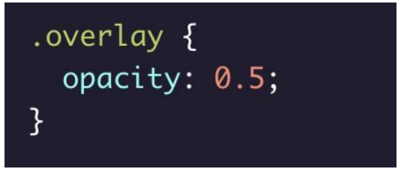
```css
h1 {
  color: red;
  background-color: blue;
}
```

In the example above, the text of the heading will appear in red, and the background of the heading will appear blue.

Opacity

Opacity is the measure of how transparent an element is. It's measured from 0 to 1, with 1 representing 100%, or fully visible and opaque, and 0 representing 0%, or fully invisible.

Opacity can be used to make elements fade into others for a nice overlay effect. To adjust the opacity of an element, the syntax looks like this:



```
.overlay {
  opacity: 0.5;
}
```

In the example above, the `.overlay` element would be 50% visible, letting whatever is positioned behind it show through.

Background Image

CSS has the ability to change the background of an element. One option is to make the background of an element an image. This is done through the CSS property `background-image`. Its syntax looks like this:

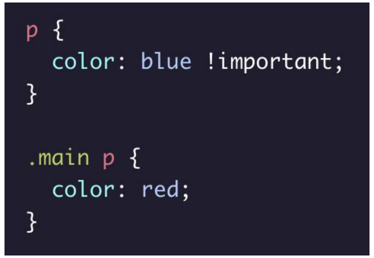
```css
.main-banner {
  background-image:
url('https://www.example.com/image.jpg');
}
```

1. The `background-image` property will set the element’s background to display an image.
2. The value provided to `background-image` is a `url`. The `url` should be a URL to an image. The `url` can be a file within your project, or it can be a link to an external site. To link to an image inside an existing project, you must provide a relative file path. If there was an image folder in the project, with an image named `mountains.jpg`, the relative file path would look like below:

```css
.main-banner {
  background-image: url('images/mountains.jpg');
}
```

Important

`!important` can be applied to specific declarations, instead of full rules. It will override *any* style no matter how specific it is. As a result, it should almost never be used. Once `!important` is used, it is very hard to override. The syntax of `!important` in CSS looks like this:



```css
p {
  color: blue !important;
}

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

Since `!important` is used on the `p` selector’s `color` attribute, all `p` elements will appear `blue`, even though there is a more specific `.main p` selector that sets the `color` attribute to `red`.

One justification for using `!important` is when working with multiple stylesheets. For example, if we are using the Bootstrap CSS framework and want to override the styles for one specific HTML element, we can use the `!important` property.