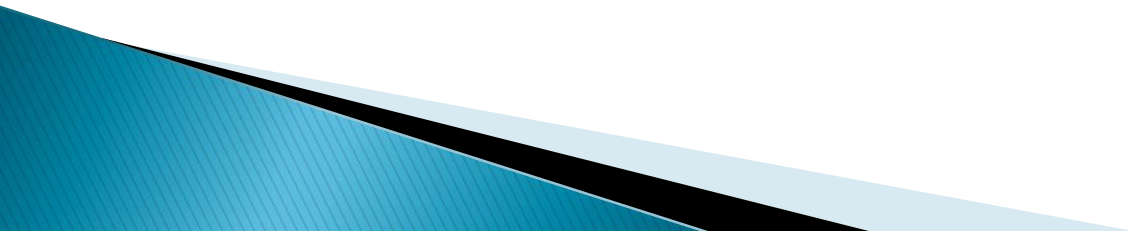


CSS In Depth

What is CSS

CSS (Cascading Style Sheets) allows you to create great-looking web pages.



HTML Document

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Getting started with CSS</title>
  </head>
  <body>
    <h1>I am a level one heading</h1>
    <p>This is a paragraph of text. In the text is a <span>span
element</span> and also a
    <a href="http://example.com">link</a>.</p>
    <p>This is the second paragraph. It contains an
    <em>emphasized</em> element.</p>
    <ul> <li>Item one</li>
    <li>Item two</li>
    <li>Item <em>three</em></li>
  </ul>
  </body>
</html>
```

Adding CSS to our Document

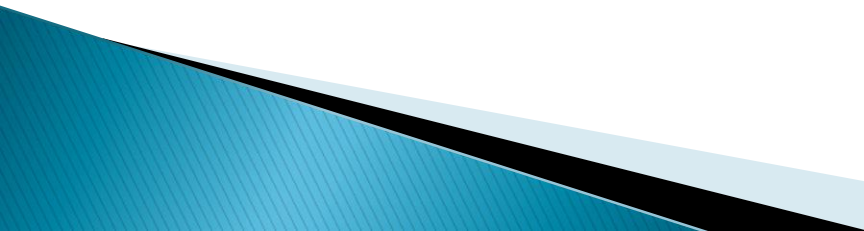
There are **three different ways to apply CSS to an HTML document** that you'll commonly come across, however, for now, we will look at the **most usual and useful way** of doing so — linking CSS from the head of your document.

Create a file in the same folder as your HTML document and save it as **styles.css**. The **.css extension shows that this is a CSS file**.

To link styles.css to index.html add the following line somewhere inside the `<head>` of the HTML document:

```
<link rel="stylesheet" href="styles.css">
```

This **<link> element** tells the browser that we have a stylesheet, using the **rel attribute**, and the **location** of that stylesheet as the value of the **href attribute**.



Styling HTML Elements

We can **style an HTML element** by **targeting an *element selector*** — this is a selector that directly matches an HTML element name.

To **turn all paragraphs green** you would use:

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

You can target **multiple selectors** at once, by **separating the selectors with a comma**.

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



Default Behaviour of an Element

Browser makes the HTML readable by adding some default styling. Headings are large and bold and our list has bullets. This happens because browsers have internal stylesheets containing default styles, which they apply to all pages by default.

Default styles can be changed by simply choosing the HTML element that you want to change, and using a CSS rule to change the way it looks.

```
li {  
list-style-type: none;  
}
```

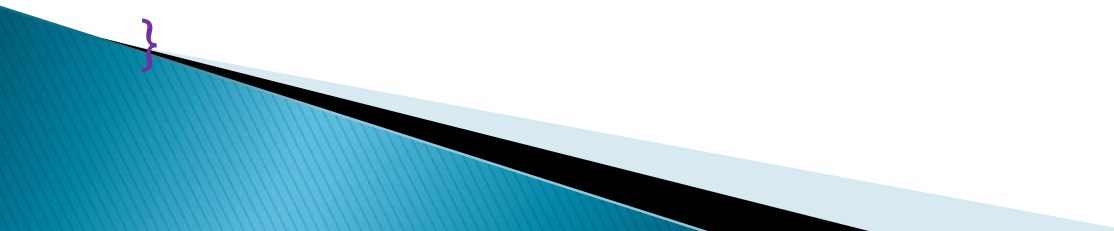
Adding Class

If you **don't want all of the elements of a type in your document to look the same** do this is to **add a class** to your HTML element and **target that class**.

```
<ul>
  <li>Item one</li>
  <li class="special">Item two</li>
  <li>Item <em>three</em></li>
</ul>
```

In your CSS you can target the class of special by creating a selector that starts with a full stop character.

```
.special {
color: orange;
font-weight: bold;
}
```



Adding Class

You can apply the class of `special` to any element on your page that you want to have the same look as this list item. For example, you might want the `` in the paragraph to also be orange and bold. Try adding a class of `special` to it.

Sometimes you will see rules with a selector that lists the HTML element selector along with the class:

```
li.special {  
  color: orange;  
  font-weight: bold;  
}
```

This syntax means "target any `li` element that has a class of `special`".



Styling based on location

Add the following rule to your Stylesheet.

```
li em {  
  color: rebeccapurple;  
}
```

This selector will select any `` element that is inside (a descendant of) an ``.

Something else you might like to try is styling a paragraph when it comes directly after a heading at the same hierarchy level in the HTML. To do so place a `+` (an **adjacent sibling combinator**) between the selectors.

```
h1 + p {  
  font-size: 200%;  
}
```

Styling Based on State

This has different states depending on whether it is unvisited, visited, being hovered over, focused via the keyboard, or in the process of being clicked (activated).

```
a:link {  
  color: pink;  
}  
a:visited {  
  color: green;  
}
```

You can change the way the link looks when the user hovers over it, for example removing the underline, which is achieved by in the next rule:

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



Applying CSS to your HTML

External Stylesheet

- ▶ This is the most common and useful method of attaching CSS to a document as you can link the CSS to multiple pages, allowing you to style them all with the same stylesheet. In most cases, the different pages of a site will all look pretty much the same, therefore you can use the same set of rules for the basic look and feel.
- ▶ An external stylesheet is when you have your CSS written in a separate file with a .css extension, and you reference it from an HTML <link> element:

Combining Selector

You can combine multiple selectors and combinators together.

/ selects any that is inside a <p>, which is inside an <article> */*

article p span { ... }

/ selects any <p> that comes directly after a , which comes directly after an <h1> */*

h1 + ul + p { ... }

Combining Selector and

You can combine multiple types together,

```
body h1 + p .special {  
  color: yellow; background-color: black; padding:  
  5px; }
```

This will style any element with a class of special, which is inside a <p>, which comes just after an <h1>, which is inside a <body>

Applying CSS to your HTML

External Stylesheet

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

Link css with HTML file:

```
<head> <meta charset="utf-8">  
<title>My CSS experiment</title>  
<link rel="stylesheet" href="styles.css">  
</head>
```

Applying CSS to your HTML

Internal Stylesheet

- ▶ An internal stylesheet is where you don't have an external CSS file, but instead place your CSS inside a `<style>` element contained inside the HTML `<head>`.
- ▶ So the HTML would look like this:

```
<head>
  <meta charset="utf-8">
  <title>My CSS experiment</title>
  <style>
    h1 {
      color: blue;
      background-color: yellow;
      border: 1px solid black;
    }
    p { color: red; }
  </style>
```

Applying CSS to your HTML

Inline styles

Inline styles are CSS declarations that **affect one element only**, contained within a style attribute:

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<meta charset="utf-8">
```

```
<title>My CSS experiment</title>
```

```
</head>
```

```
<body>
```

```
<h1 style="color: blue;background-color: yellow;border: 1 px  
solid black;">Hello World!</h1>
```

```
<p style="color:red;">This is my first CSS example</p>
```

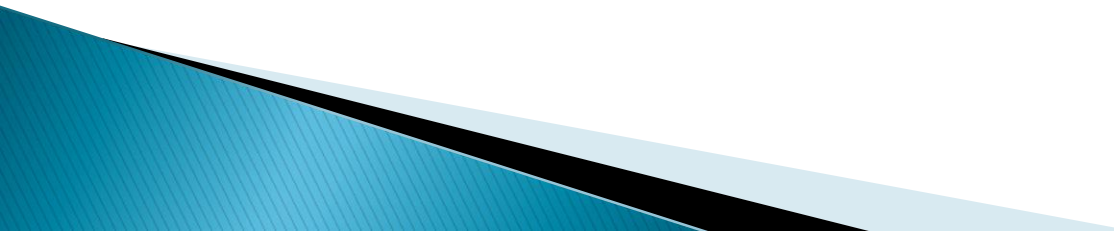
```
</body>
```

```
</html>
```

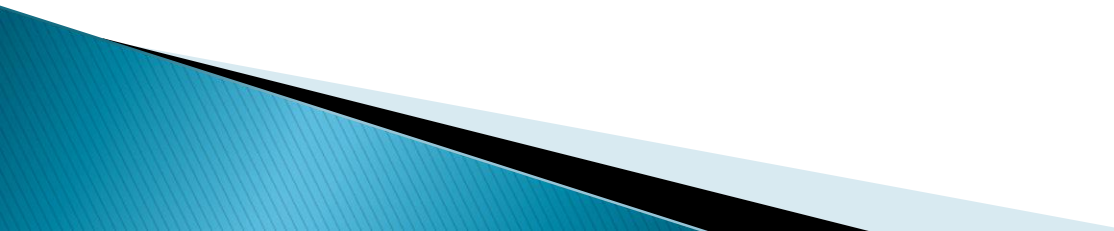

Applying CSS to your HTML

Inline styles

Please don't do this, unless you really have to! It is really bad for maintenance (you might have to update the same information multiple times per document), and **it also mixes your presentational CSS information with your HTML structural information, making the code harder to read and understand.** Keeping different types of code separated makes for a much easier job for all who work on the code.



Properties and Values

- ▶ **Properties:** Human-readable identifiers that indicate which stylistic features (e.g. font-size, 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.)
- 

Properties and Values

There will often be scenarios where **two selectors could select the same HTML element**. Consider the stylesheet below where I have a rule with a p selector that will set paragraphs to blue, and also a class that will set selected elements red.

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

```
p { color: blue; }
```

This is the **cascade in action**.

```
.special { color: red; }
```

```
p { color: blue; }
```

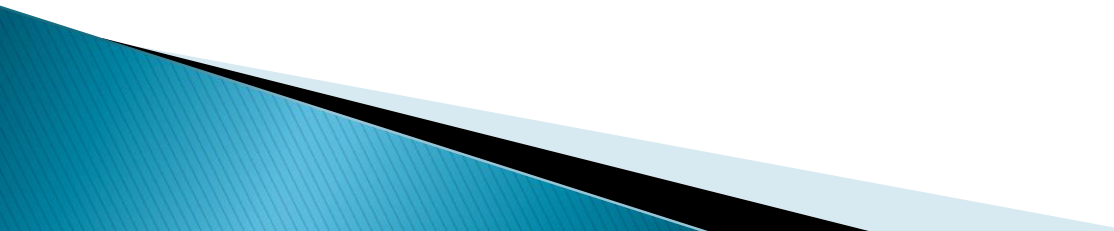
HTML

```
<p class="special">What color am I?</p>
```

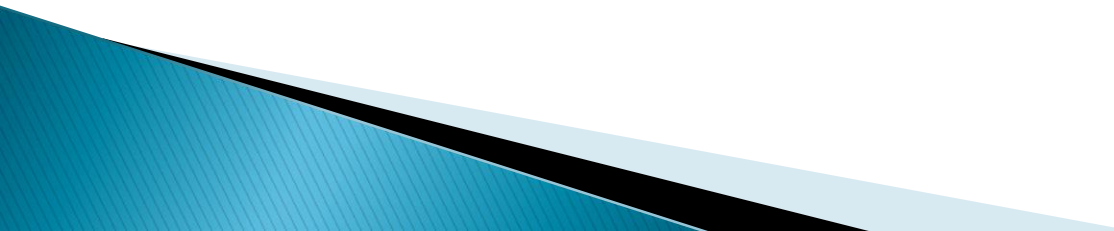
Properties and Values

Important: If a property is unknown or if a value is not valid for a given property, the declaration is deemed *invalid* and is completely 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 language uncertainty arises. For example, color should *always* be spelled color. colour won't work.



CSS Selectors

- ▶ `h1`
 - ▶ `a:link`
 - ▶ `.manythings`
 - ▶ `#onething`
 - ▶ `*`
 - ▶ `.box p`
 - ▶ `.box p:first-child`
 - ▶ `h1, h2, .intro`
- 

Specificity

There will often be scenarios where **two selectors could select the same HTML element**. Consider the stylesheet below where I have a rule with **a p selector** that will set paragraphs to blue, and also a class that will set selected elements red.

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

```
.special { color: red; }  
p { color: blue; }
```

A class is described as being **more specific**, or **having more specificity than the element** selector, so it wins.



Functions

There are **some possible values** which **take the form of a function**. An example would be the **calc() function**. This function allows you to do simple math from within your CSS, for example:

```
<div class="outer"><div class="box">The inner box  
is 90% – 30px.</div></div>
```

```
.outer { border: 5px solid black; }  
.box {  
padding: 10px;  
width: calc(90% – 30px);  
background-color: rebeccapurple;  
color: white; }
```

HOW CSS VALUES ARE PROCESSED

	width (paragraph)	padding (paragraph)	font-size (root)	font-size (section)	font-size (paragraph)
1. Declared value (author declarations)	140px 66%	–	–	1.5rem	–
2. Cascaded value (after the cascade)	66%	–	16px (Browser default)	1.5rem	–
3. Specified value (defaulting, if there is no cascaded value)	66%	0px (Initial value)	16px	1.5rem	24px
4. Computed value (converting relative values to absolute)	66%	0px	16px	24px (1.5 * 16px)	24px
5. Used value (final calculations, based on layout)	184.8px	0px	16px	24px	24px
6. Actual value (browser and device restrictions)	185px	0px	16px	24px	24px

```
<div class="section">
  <p class="amazing">CSS is absolutely amazing</p>
</div>
```

```
.section {
  font-size: 1.5rem;
  width: 280px;
  background-color: orangered;
}

p {
  width: 140px;
  background-color: green;
}

.amazing {
  width: 66%;
}
```

CSS is absolutely amazing

(Let's analyse the green paragraph)

uberry

HOW UNITS ARE CONVERTED FROM RELATIVE TO ABSOLUTE (PX)

	Example (x)		How to convert to pixels		Result in pixels
Font-based	% (fonts)	150%	→	$x\% \times \text{parent's computed font-size}$	→ 24px
	% (lengths)	10%	→	$x\% \times \text{parent's computed width}$	→ 100px
	em (font)	3em	→	$x \times \text{parent computed font-size}$	→ 72px (3 * 24)
	em (lengths)	2em	→	$x \times \text{current element computed font-size}$	→ 48px
	rem	10rem	→	$x \times \text{root computed font-size}$	→ 160px
Viewport-based	vh	90vh	→	$x \times 1\% \text{ of viewport height}$	→ 90% of the current viewport height
	vw	80vw	→	$x \times 1\% \text{ of viewport width}$	→ 80% of the current viewport width

```
html, body {  
  font-size: 16px;  
  width: 80vw;  
}  
  
header {  
  font-size: 150%;  
  padding: 2em;  
  margin-bottom: 10rem;  
  height: 90vh;  
  width: 1000px;  
}  
  
.header-child {  
  font-size: 3em;  
  padding: 10%;  
}
```

CSS VALUE PROCESSING: WHAT YOU NEED TO KNOW

- Each property has an initial value, used if nothing is declared (and if there is no inheritance — see next lecture);
- Browsers specify a **root font-size** for each page (usually 16px);
- Percentages and relative values are always converted to pixels;
- Percentages are measured relative to their parent's **font-size**, if used to specify font-size;
- Percentages are measured relative to their parent's **width**, if used to specify lengths;
- **em** are measured relative to their **parent** font-size, if used to specify font-size;
- **em** are measured relative to the **current** font-size, if used to specify lengths;
- **rem** are always measured relative to the **document's root** font-size;
- **vh** and **vw** are simply percentage measurements of the viewport's height and width.

INHERITANCE: WHAT YOU NEED TO KNOW

- Inheritance passes the values for some specific properties from parents to children — **more maintainable code**;
- Properties related to text are inherited: `font-family`, `font-size`, `color`, etc;
- The computed value of a property is what gets inherited, **not** the declared value.
- Inheritance of a property only works if no one declares a value for that property;
- The `inherit` keyword forces inheritance on a certain property;
- The `initial` keyword resets a property to its initial value.

2. BOX TYPES: INLINE, BLOCK-LEVEL AND INLINE-BLOCK

Block-level boxes

- Elements formatted visually as blocks
- 100% of parent's width
- Vertically, one after another
- Box-model applies as showed

`display: block`

`(display: flex)`
`(display: list-item)`
`(display: table)`

Inline-block boxes

- A mix of block and inline
- Occupies only content's space
- No line-breaks
- Box-model applies as showed

`display: inline-block`

Inline boxes

- Content is distributed in lines
- Occupies only content's space
- No line-breaks
- No heights and widths
- Paddings and margins only horizontal (left and right)

`display: inline`

3. POSITIONING SCHEMES: NORMAL FLOW, ABSOLUTE POSITIONING AND FLOATS

Normal flow

- Default positioning scheme;
- **NOT** floated;
- **NOT** absolutely positioned;
- Elements laid out according to their source order.

Default
`position: relative`

Floats

- **Element is removed from the normal flow;**
- Text and inline elements will wrap around the floated element;
- The container will not adjust its height to the element.

`float: left`
`float: right`

Absolute positioning

- **Element is removed from the normal flow**
- No impact on surrounding content or elements;
- We use top, bottom, left and right to offset the element from its relatively positioned container.

`position: absolute`
`position: fixed`

=

≠

4. STACKING CONTEXTS

