Introduction to CSS

What is CSS?

CSS (Cascading Style Sheets) is a language that describes the style of an HTML document. CSS describes how HTML elements should be displayed.

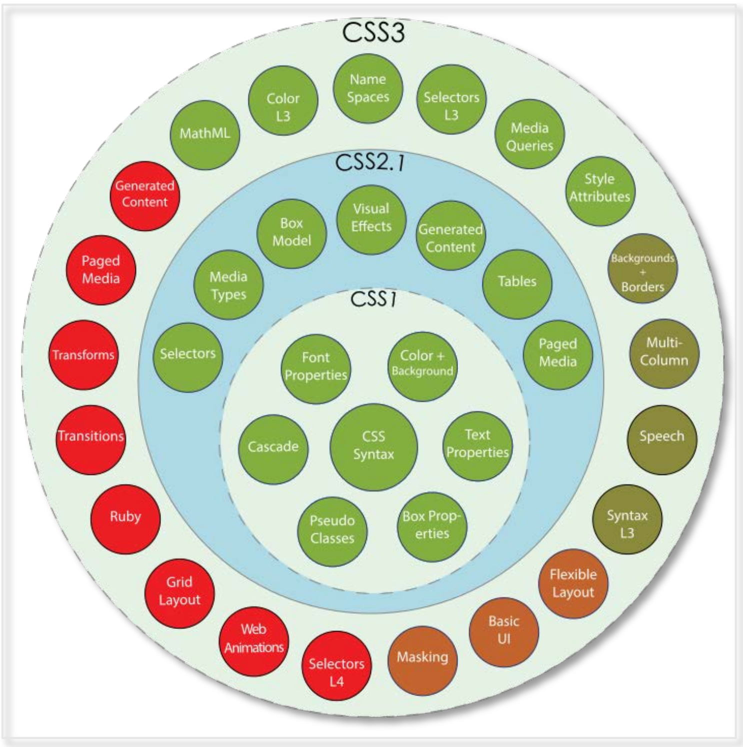
- CSS stands for Cascading Style Sheets.
- CSS describes how HTML elements are to be displayed on screen, paper, or in other media.
- CSS saves a lot of work. It can control the layout of multiple web pages all at once.
- External stylesheets are stored in CSS files.

History

CSS was first developed in 1997 as a way for web developers to define the visual appearance of the web pages that they were creating. It was intended to allow web professionals to separate the content and structure of a website's code from the visual design, something that had not been possible prior to this time.

The separation of structure and style allows HTML to perform more of its original function, the markup of content, without having to worry about the design and layout of the page itself, something commonly known as the "look and feel" of the page.

CSS didn't gain in popularity until around 2000 when web browsers began using more than the basic font and colour aspects of this markup language. As CSS continues to evolve and new styles are introduced, web browsers have begun to implement modules that bring new CSS support into those browsers and give web designers powerful new styling tools to work with.



The Schematic of the evolution of CSS from 1 to 3

https://goo.gl/2cwT1I

Background

HTML was NEVER intended to contain tags for formatting a web page. HTML was created to describe the content of a web page, like:

```
<h1>This is a heading</h1>
<p>This is a paragraph.</p>
```

When tags like <font>, and colour attributes were added to the HTML 3.2 specification, it created huge problems for web developers. Development of large websites, where fonts and colour information were added to every single page, became a long and expensive process. To solve this problem, the World Wide Web Consortium (W3C) created CSS. CSS removed the style formatting from the HTML page.

The style definitions are normally saved in external .css files. With an external stylesheet file, you can change the look of an entire website by changing just one file!

CSS Example

The CSS Zen Garden is a World Wide Web development resource "built to demonstrate what can be accomplished visually through CSS-based design." Style sheets contributed by designers from around the world are used to change the visual presentation of a single HTML file, producing hundreds of different designs.

Explore the CSS Zen Garden example files which are supplied - HTML and CSS file. Visit the CSS zen website to view different designs. Clicking on any one will load the style sheet into this very page. The HTML remains the same, the only thing that has changed is the external CSS file. A demonstration of what can be accomplished through CSS-based design.

http://www.csszengarden.com/

Reflection

Reflect on what you have learned about CSS so far.

Use the space below to write **five** things about CSS.

1

2

3

4

5

CSS Structure

Structure

CSS is a language for specifying how documents are presented to users - how they are styled, laid out, etc.

A document is usually a text file structured using a markup language - HTML is the most common markup language, but you will also come across other markup languages such as SVG or XML.

Presenting a document to a user means converting it into a usable form for your audience. Browsers, like Firefox, Chrome or Internet Explorer, are designed to present documents visually, for example, on a computer screen, projector or printer.

How does CSS affect HTML?

Web browsers apply CSS rules to a document to affect how they are displayed. A CSS rule is formed from:

- A set of properties, which have values set to update how the HTML content is displayed, for example, the element's width is 50% of its parent element, and its background to be red.

- A selector, which selects the element(s) you want to apply the updated property values to. For example, the application a CSS rule to all the paragraphs in a HTML document.

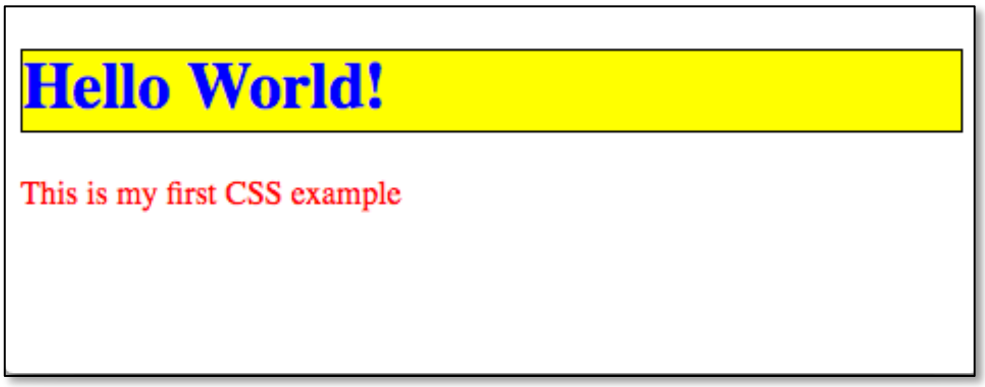
A set of CSS rules contained within a stylesheet determines how a webpage should look.

CSS Example II

Open the two files for this exercise in the folder called Structure. The first rule starts with an h1 selector, which means that it will apply its property values to the <h1> element. It contains three properties and their values.

1. The first one sets the text colour to blue.
2. The second sets the background colour to yellow.
3. The third one puts a border around the header that is 1 pixel wide, solid (not dotted, or dashed, etc.), and coloured black.

The second rule starts with a p selector, which means that it will apply its property values to the <p> element. It contains one declaration, which sets the text colour to red.
In a web browser, the code above would produce the following output:



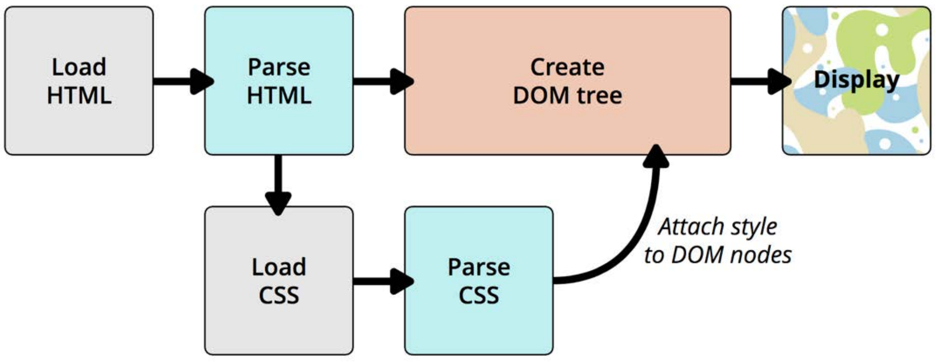
```
Hello World!

This is my first CSS example
```

How does CSS work?

When a browser displays a document, it must combine the document's content with its style information. It processes the document in two stages:

1. The browser converts HTML and CSS into the DOM (*Document Object Model*). The DOM represents the document in the computer's memory. It combines the document's content with its style.

2. The browser displays the contents of the DOM.



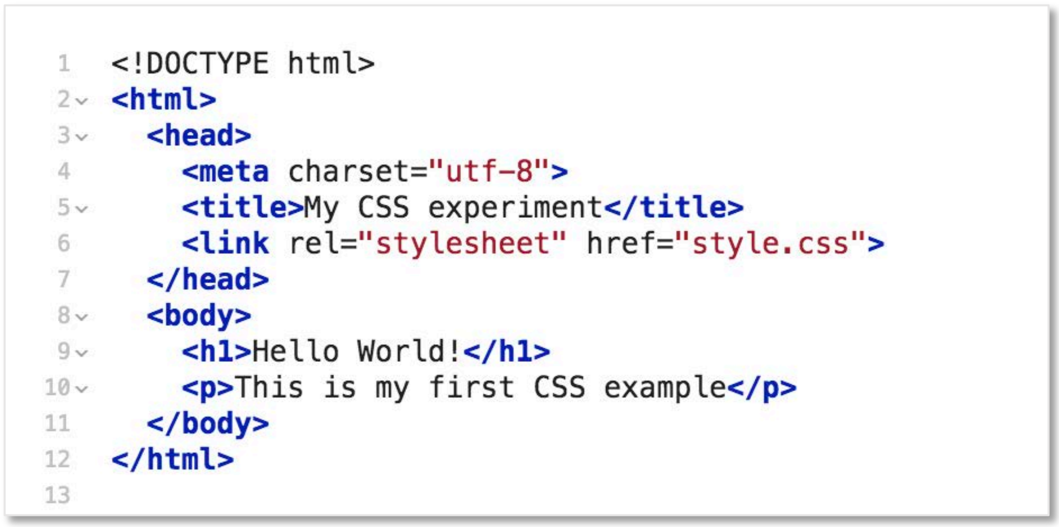
How to apply your CSS to your HTML

There are three different ways to apply CSS to an HTML document that you'll commonly come across, some more useful than others. There are three ways of inserting a style sheet:

- External style sheet
- Internal style sheet
- Inline style

External stylesheet

You've already seen external stylesheets previously, but not by that name. 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. The HTML file looks something like this:



```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>
  </body>
</html>
```

And the CSS file:

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

p {
  color: red;
}

```

This method is arguably the best, as you can use one stylesheet to style multiple documents.

Internal stylesheet

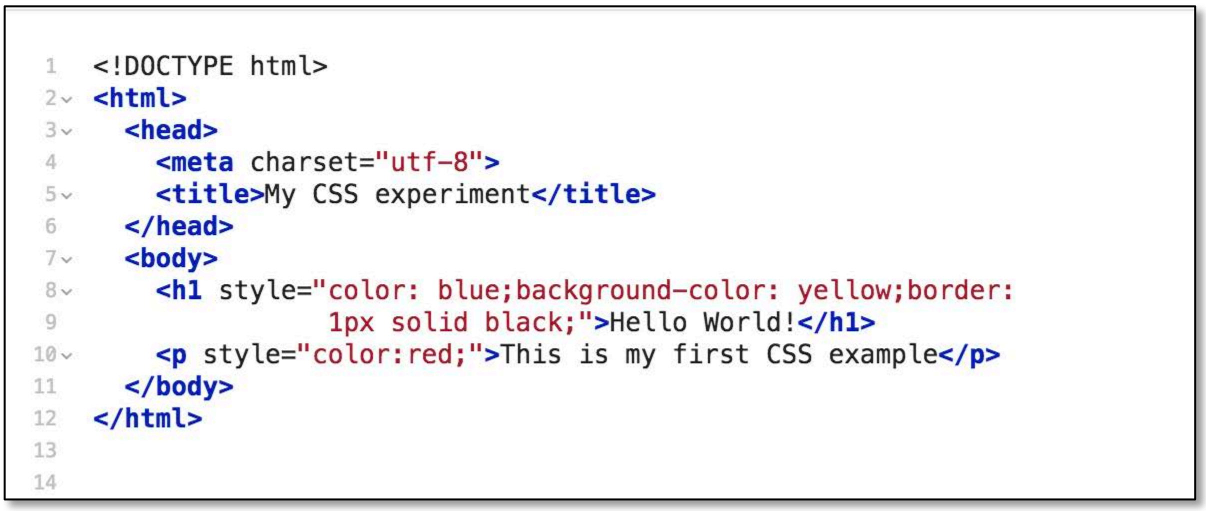
An internal stylesheet is where you don't have an external CSS file, but instead place your CSS inside a <style> element, contained inside the HTML head. So the HTML would look like this:

```
<!DOCTYPE html>
<html>
  <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>
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>
  </body>
</html>

```

This can be useful in some circumstances (maybe you're working with a content management system where you can't modify the CSS files directly), but it isn't quite as efficient as external stylesheets - in a website, the CSS would need to be repeated across every page.

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:
              1px solid black;">Hello World!</h1>
    <p style="color:red;">This is my first CSS example</p>
  </body>
</html>
```

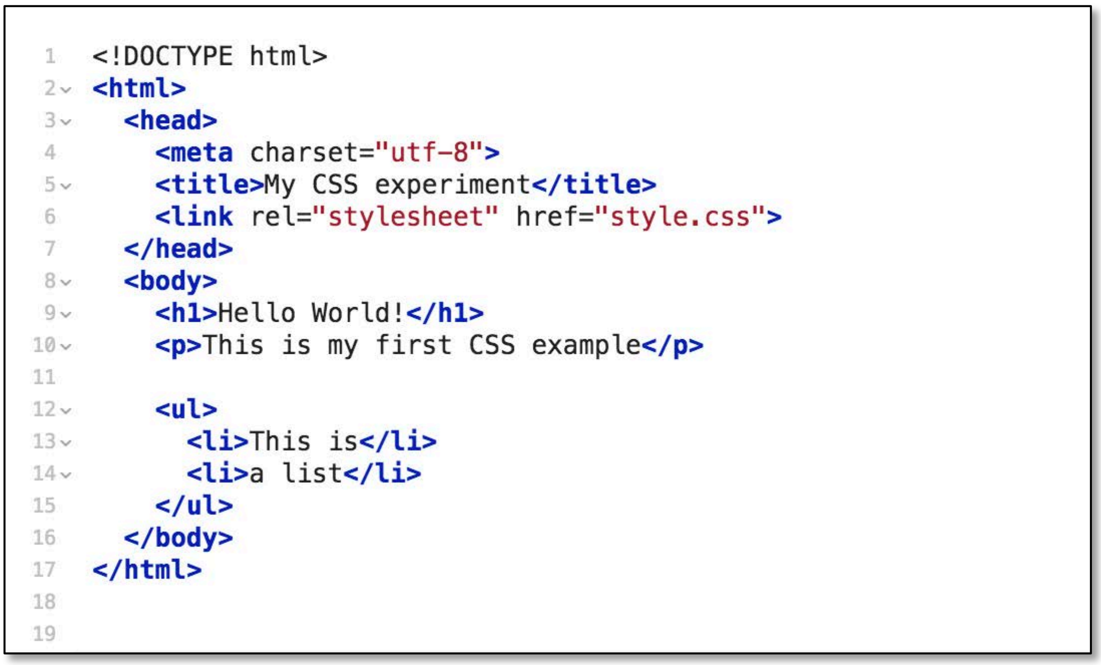
The only time you might have to resort to using inline styles is when your working environment is really restrictive (perhaps your CMS only allows you to edit the HTML body).

CSS syntax

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

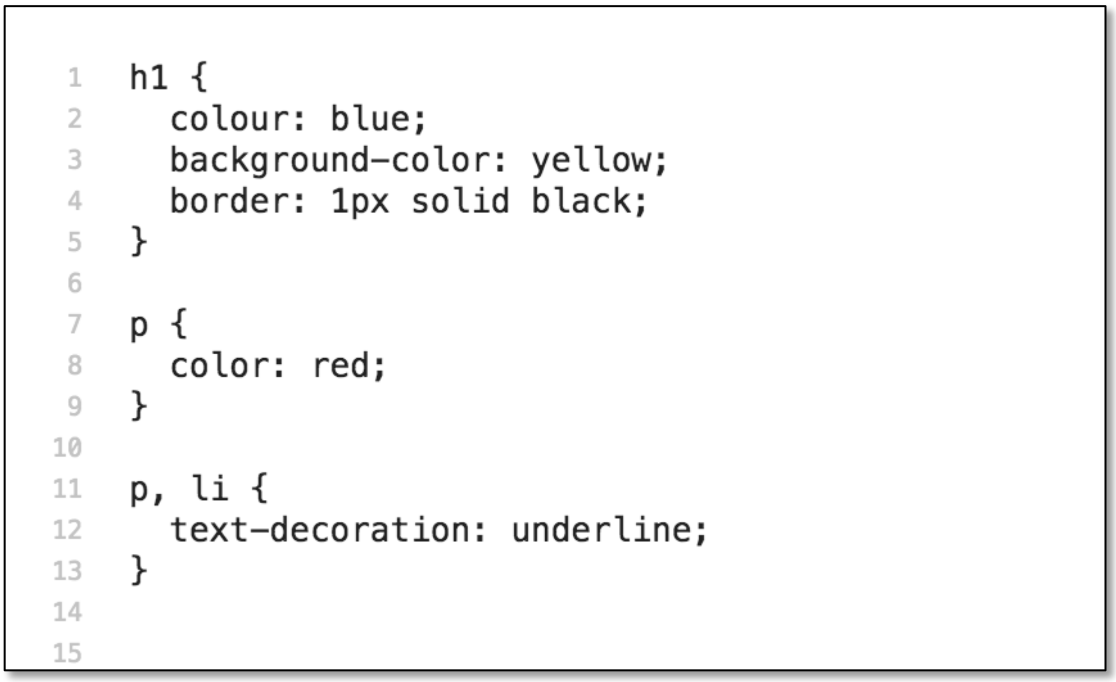
> **Properties**: Human-readable identifiers that indicate which stylistic features (e.g. font, width, background colour) you want to change.
>
> **Values**: Each specified property is given a value, which indicates how you want to change those stylistic features (e.g. the font, width or background colour).

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

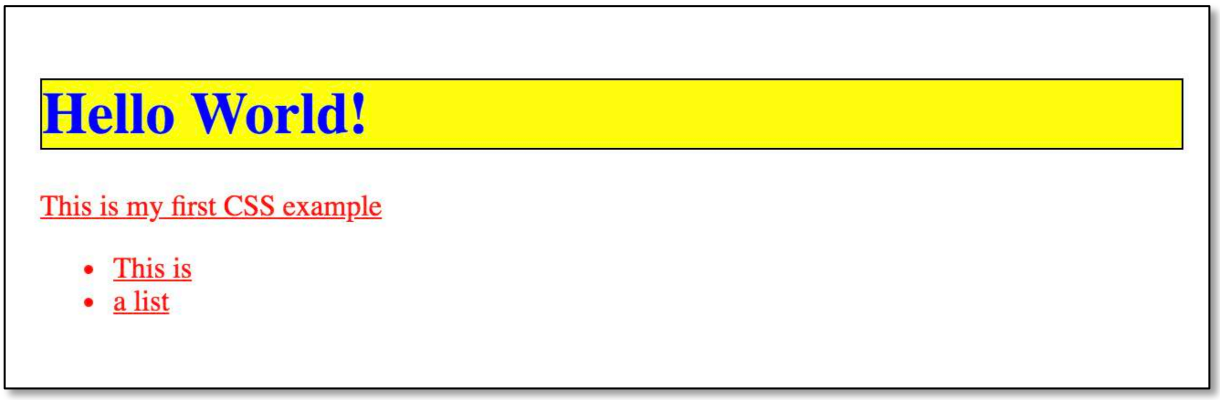


```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>My CSS experiment</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <h1>Hello World!</h1>
    <p>This is my first CSS example</p>

    <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;
}

p, li {
  text-decoration: underline;
}
```



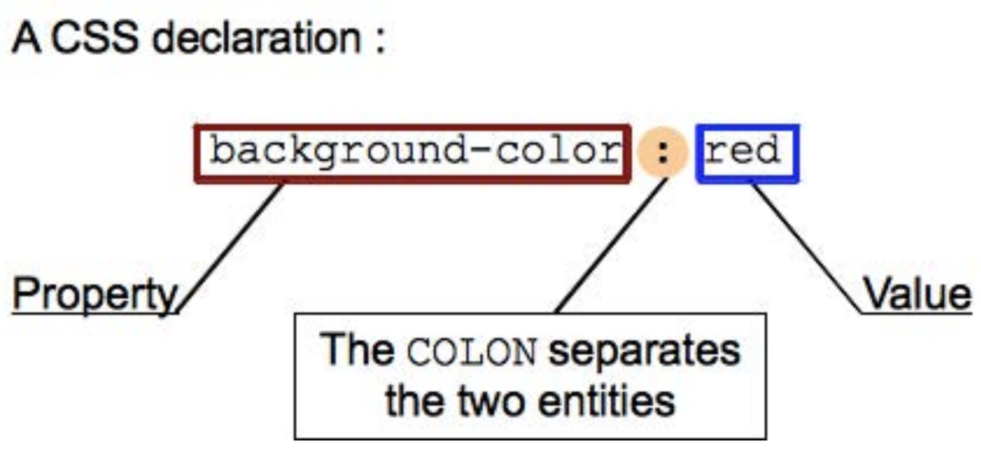
Hello World!

This is my first CSS example

- This is
- a list

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. 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.

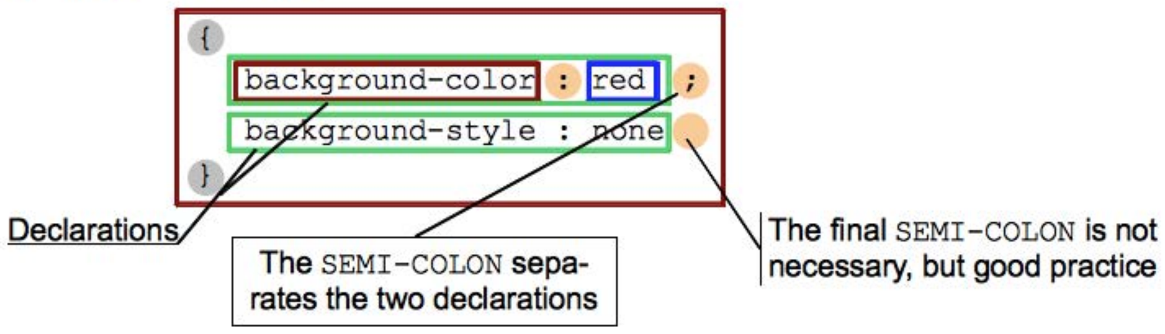


CSS declaration blocks

Declarations are grouped in blocks, with each set of declarations being wrapped by opening curly brackets { 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.

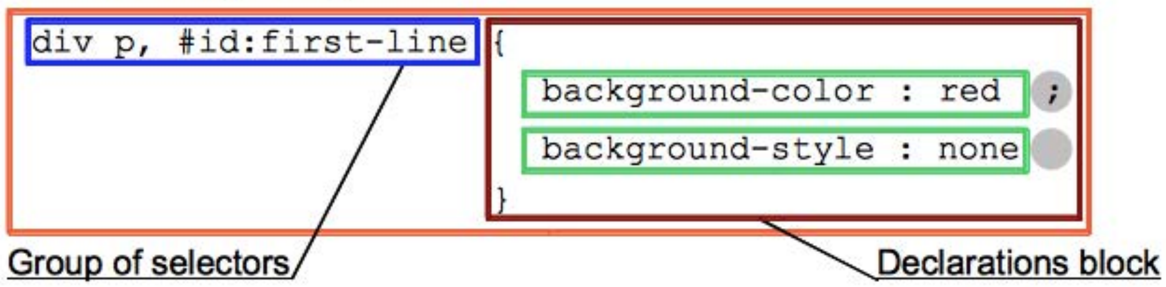
A CSS declarations block:


{
background-color
:
red
;
background-style : none
}
Declarations
The SEMI-COLON separates the two declarations
The final SEMI-COLON is not necessary, but good practice

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 a rule.

A CSS ruleset (or rule):



Beyond syntax: make CSS readable

There are some good tips 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.

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 */.

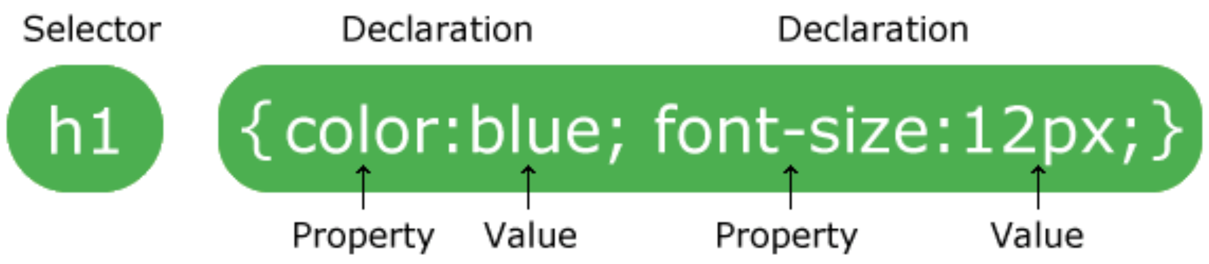
Reflection

Reflect on what you have learned about how CSS works.

Use the space below

Introduction to Selectors

To recap, selectors are one part of a CSS rule and come just before CSS declaration blocks.



Different types of selectors

Selectors can be divided into the following categories:

- Simple selectors: Match one or more elements based on element type, class, or id.
- Attribute selectors: Match one or more elements based on their attributes/attribute values.
- Pseudo-classes: Match one or more elements that exist in a certain state, such as an element that is being hovered over by the mouse pointer, or a checkbox that is currently disabled or checked, or an element that is the first child of its parent in the DOM tree.
- Pseudo-elements: Match one or more parts of content that are in a certain position in relation to an element, for example the first word of each paragraph, or generated content appearing just before an element.
- Combinators: These are not exactly selectors themselves, but ways of combining two or more selectors in useful ways for very specific selections. So for example, you could select only paragraphs that are direct descendants of divs, or paragraphs that come directly after headings.

- Multiple selectors: Again, these are not separate selectors; the idea is that you can put multiple selectors on the same CSS rule, separated by commas, to apply a single set of declarations to all the elements selected by those selectors.

Simple selectors

Type selectors /element selectors

This selector is just a case-insensitive match between the selector name and a given HTML element name. This is the simplest way to target all elements of a given type. Let's take a look at an example:

View the example above (HTML & CSS) in folder supplied.

Class selectors

The class selector consists of a dot, '.', followed by a class name. A class name is any value, without spaces, placed within an HTML class attribute. It is up to you to choose a name for the class. It is also noteworthy that multiple elements in a document can have the same class value, and a single element can have multiple class names separated by white space.

View the example above (HTML & CSS) in folder supplied.

ID selectors

The ID selector consists of a hash/pound symbol (#), followed by the ID name of a given element. Any element can have a unique ID name set with the id attribute. It is up to you to choose an ID name. It's the most efficient way to select a single element.

View the example above (HTML & CSS) in folder supplied.

The CSS Layout Workshop

CSS Selectors

Rachel Andrew | thecssworkshop.com

CSS Selectors Video - https://youtu.be/viJJoo8uJuY

Pseudo-classes and pseudo-elements

Pseudo-classes

A CSS pseudo-class is a keyword added to the end of a selector, preceded by a colon (:), which is used to specify that you want to style the selected element but only when it is in a certain state. For example, you might want to style a link element only when it is being hovered over by the mouse pointer, or a checkbox when it is disabled or checked. For example:

- :active
- :checked
- :default
- :dir
- :disabled
- :empty
- :enabled

- :first
- :first-child

We will look into every pseudo-class right now but here is a simple example.

Mozilla Developer Network

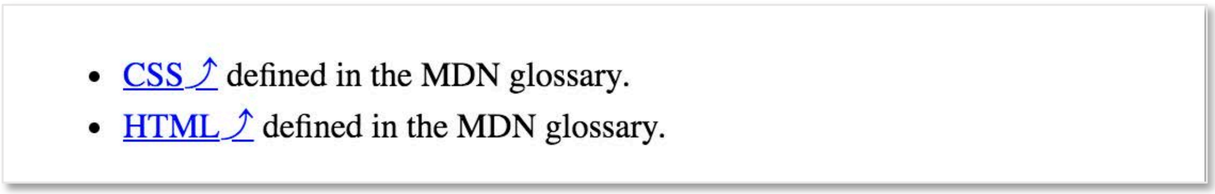
View the example files (HTML & CSS) in folder supplied.

Pseudo-elements

Pseudo-elements are very much like pseudo-classes, but they have differences. They are keywords, this time preceded by two colons :: , that can be added to the end of selectors to select a certain part of an element.

- ::after
- ::before
- ::first-letter
- ::first-line
- ::selection
- ::backdrop

They all have some very specific behaviours and interesting features.


- CSS ⤴ defined in the MDN glossary.
- HTML ⤴ defined in the MDN glossary.

View the example above (HTML & CSS) in folder supplied.

CSS Units

- CSS has several different units for expressing a length.
- Many CSS properties take "length" values, such as width, margin, padding, font-size, etc.
- Length is a number followed by a length unit, such as 10px, 2em, etc.
- A whitespace cannot appear between the number and the unit. However, if the value is 0, the unit can be omitted.
- For some CSS properties, negative lengths are allowed.
- There are two types of length units: absolute and relative.

Absolute Lengths

The absolute length units are fixed and a length expressed in any of these will appear as exactly that size.

Absolute length units are not recommended for use on screen, because screen sizes vary so much. However, they can be used if the output medium is known, such as for print layout.

Unit	Description
cm	centimetres
mm	millimetres
in	inches (1in = 96px = 2.54cm)
px *	pixels (1px = 1/96th of 1in)

pt	points (1pt = 1/72 of 1in)
pc	picas (1pc = 12 pt)

Relative Lengths

Relative length units specify a length relative to another length property. Relative length units scales better between different rendering mediums.

Unit	Description
em	Relative to the font-size of the element (2em means 2 times the size of the current font)
ex	Relative to the x-height of the current font (rarely used)
ch	Relative to width of the "0" (zero)
rem	Relative to font-size of the root element
vw	Relative to 1% of the width of the viewport*
vh	Relative to 1% of the height of the viewport*
vmin	Relative to 1% of viewport's* smaller dimension
vmax	Relative to 1% of viewport's* larger dimension
%	Relative to the parent element

Reflection

Reflect on what you have learned about CSS units.

Use the space below

Cascade

At some point in your work, you'll find yourself in the situation where multiple CSS rules will have selectors matching the same element. In such cases, which CSS rule "wins", and ends up being the one that is finally applied to the element? This is controlled by a mechanism called the *Cascade*; this is also related to inheritance (elements will take some property values from their parents, but not others).

The Cascade

CSS is an abbreviation for *Cascading Style Sheets*, which indicates that the notion of the cascade is important. At its most basic level, it indicates that the order of CSS rules matter, but it's more complex than that. What selectors win out in the cascade depends on three factors (these are listed in order of weight - earlier ones will overrule later ones):

- Importance
- Specificity
- Source order

Importance

In CSS, there is a special piece of syntax you can use to make sure that a certain declaration will always win over all others: *!important.*

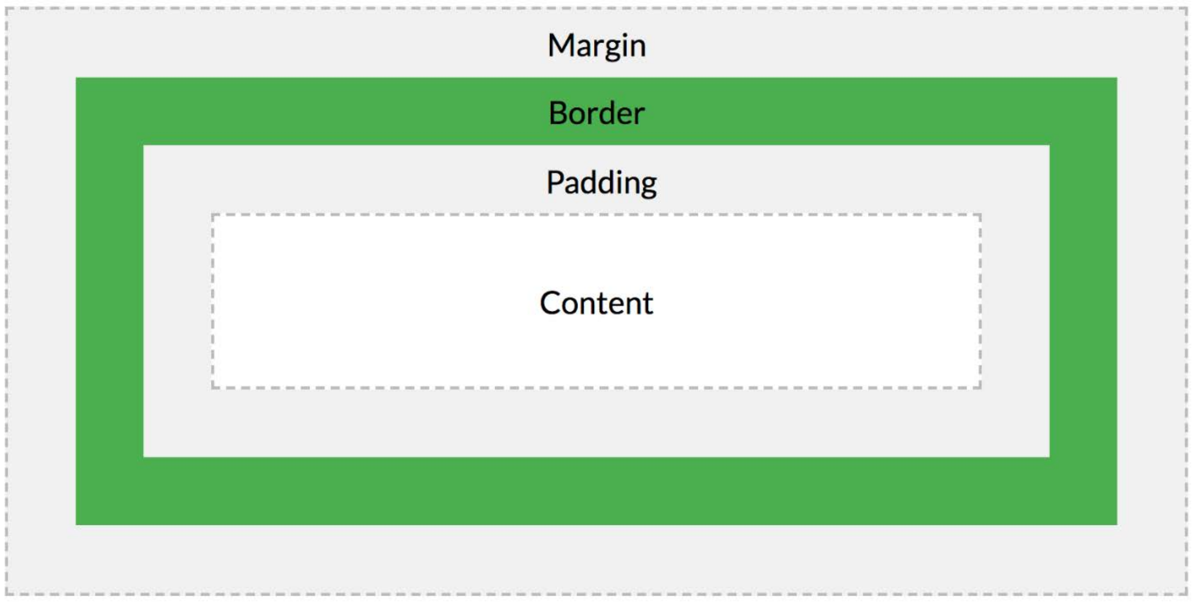
TEACHER TIP
To learn more on Cascade, please see visit https://developer.mozilla.org/en-US/docs/Web/CSS/Cascade

The Box Model

The CSS box model is the foundation of layout on the Web - each element is represented as a rectangular box, with the box's content, padding, border, and margin built up around one another like the layers of an onion. As a browser renders the web page layout, it works out what styles are applied to the content of each box, how big the surrounding onion layers are, and where the boxes sit in relation to one another. Before understanding how to create CSS layouts, you need to understand the box model.

All HTML elements can be considered as boxes. In CSS, the term "box model" is used when talking about design and layout.

The CSS box model is essentially a box that wraps around every HTML element. It consists of: margins, borders, padding, and the actual content. The image below illustrates the box model:



Explanation of the different parts:

- **Content** - The content of the box, where text and images appear.
- **Padding** - Clears an area around the content. The padding is transparent.
- **Border** - A border that goes around the padding and content.
- **Margin** - Clears an area outside the border. The margin is transparent.

The box model allows us to add a border around elements, and to define space between elements.

TEACHER TIP
The default behaviour of browsers when calculating the width of an element is to apply the calculated width and height to the content area, without taking any of the padding, border and margin in consideration. This approach has proven to be quite complicated to work with. You can change this behaviour by setting the box-sizing property.

TEACHER TIP
Use the W3C CSS validator to validate your CSS - https://jigsaw.w3.org/css-validator

Styling Text

As you'll have already experienced in your work with HTML and CSS, text inside an element is laid out inside the element's content box. It starts at the top left of the content area, and flows towards the end of the line. Once it reaches the end, it goes down to the next line and continues, then the next line, until all the content has been placed in the box. Text content effectively behaves like a series of inline elements, being laid out on lines adjacent to one another, and not creating line breaks until the end of the line is reached, or unless you force a line break manually using the <br> element.

The CSS properties used to style text generally fall into two categories, which we'll look at separately:

Font styles: Properties that affect the font that is applied to the text, affecting what font is applied, how big it is, whether it is bold, italic, etc.

Text layout styles: Properties that affect the spacing and other layout features of the text, allowing manipulation of, for example, the space between lines and letters, and how the text is aligned within the content box.

Font Properties

The CSS font properties define the font family, weight, size, and the style of a text**.**

The font-family Property

The font-family property specifies the font for an element. The font-family property can hold several font names as a second choice. If the browser does not support the first font, it tries the next font.

There are two types of font family names:

- **family-name -** The name of a font-family, like "times", "courier", "arial", etc.
- **generic-family -** The name of a generic-family, like "serif", "sans-serif", "cursive", "fantasy", "monospace".

Start with the font you want, and always end with a generic family, to let the browser pick a similar font in the generic family, if no other fonts are available.

TEACHER TIP
Separate each value with a comma.

If a font name contains white-space, it must be quoted. Single quotes must be used when using the "style" attribute in HTML .

Font style, font weight, text transform, and text decoration

CSS provides four common properties to alter the visual weight/emphasis of text:

- **font-style**: Used to turn italic text on and off. Possible values are as follows (you'll rarely use this, unless you want to turn some italic styling off for some reason):
  - normal: Sets the text to the normal font (turns existing italics off.)
  - italic: Sets the text to use the *italic version of the font* if available; if not available, it will simulate italics with oblique instead.
  - oblique: Sets the text to use a simulated version of an italic font, created by *slanting the normal version*.

- **font-weight:** Sets how bold the text is. This has many values available in case you have many font variants available (such as *-light*, *-normal*, *-bold*, *-extrabold*, *-black*, etc.), but realistically you'll rarely use any of them except for normal and bold:
  - normal, bold: Normal and **bold** font weight
  - lighter, bolder: Sets the current element's boldness to be one step lighter or heavier than its parent element's boldness.
  - 100–900: Numeric boldness values that provide finer grained control than the above keywords, if needed.

- **text-transform:** Allows you to set your font to be transformed. Values include:
  - none: Prevents any transformation.
  - uppercase: Transforms ALL TEXT TO CAPITALS.
  - lowercase: Transforms all text to lower case.
  - capitalise: Transforms all words to Have The First Letter Capitalised.

- full-width: Transforms all glyphs to be written inside a fixed-width square, similar to a monospace font, allowing aligning of e.g. Latin characters along with Asian language glyphs (like Chinese, Japanese, Korean.)

- **text-decoration**: Sets/unsets text decorations on fonts (you'll mainly use this to unset the default underline on links when styling them.) Available values are:
  - none: unsets any text decorations already present.
  - underline: Underlines the text.
  - overline: Gives the text an overline.
  - line-through: Puts a strikethrough over the text.

- **The font-size Property:** The font-size property sets the size of a font.

Text Properties

The color Property

The colour property specifies the colour of text. Use a background colour combined with a text colour that makes the text easy to read.

The text-align Property

The text-align property specifies the horizontal alignment of text in an element.

The vertical-align Property

The vertical-align property sets the vertical alignment of an element.

alue	Description
baseline	The element is aligned with the baseline of the parent. This is default
length	Raises or lowers an element by the specified length. Negative values are allowed.
%	Raises or lowers an element in a percent of the "line-height" property. Negative values are allowed
sub	The element is aligned with the subscript baseline of the parent
super	The element is aligned with the superscript baseline of the parent
top	The element is aligned with the top of the tallest element on the line
text-top	The element is aligned with the top of the parent element's font
middle	The element is placed in the middle of the parent element
bottom	The element is aligned with the lowest element on the line
text-bottom	The element is aligned with the bottom of the parent element's font

initial	**Sets this property to its default value**
inherit	**Inherits this property from its parent element**

Text layout

With basic font properties out the way, let's now have a look at properties we can use to affect text layout.

Text alignment

The text-align property is used to control how text is aligned within its containing content box. The available values are as follows, and work in pretty much the same way as they do in a regular word processor application:

- left: Left justifies the text.
- right: Right justifies the text.
- center: Centers the text.
- justify: Makes the text spread out, varying the gaps in between the words so that all lines of text are the same width. You need to use this carefully — it can look terrible, especially when applied to a paragraph with lots of long words in it. If you are going to use this, you should also think about using something else along with it, such as hyphens, to break some of the longer words across lines.

Line height

The line-height property sets the height of each line of text — this can take most length and size units, but can also take a unitless value, which acts as a multiplier and is generally considered the best option — the font-size is multiplied to get the line-height. Body text generally looks nicer and is easier to read when the lines are spaced apart; the recommended line height is around 1.5–2 (double spaced.) So to set our lines of text to 1.5 times the height of the font, you'd use this:

Reflection

Reflect on what you have learned about CSS so far.

Use the space below to write **five** things about styling text:

1.

2.

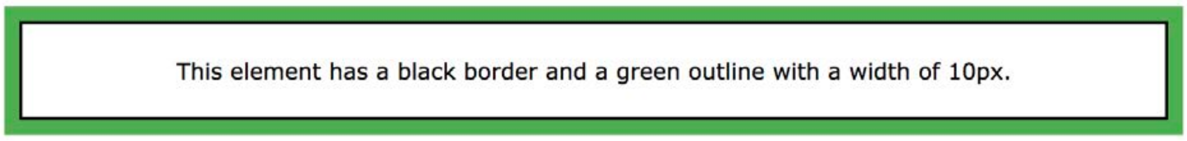
3.

4.

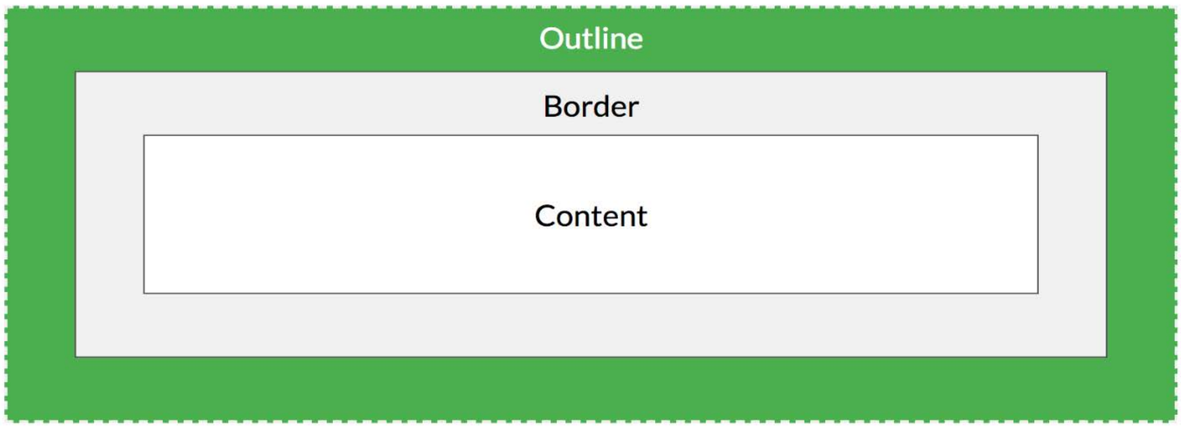
5.

Styling boxes

CSS Outline


This element has a black border and a green outline with a width of 10px.

An outline is a line that is drawn around elements, OUTSIDE the borders, to make the element "stand out".



Styling boxes overview

CSS has the following outline properties:

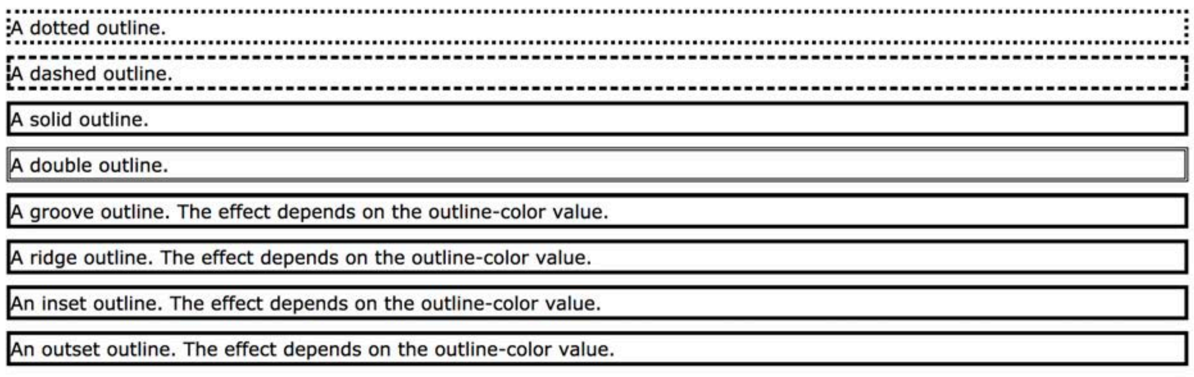
- outline-style
- outline-color
- outline-width
- outline-offset
- outline

Outline Style

The outline-style property specifies the style of the outline, and can have one of the following values:

- dotted - Defines a dotted outline
- dashed - Defines a dashed outline
- solid - Defines a solid outline
- double - Defines a double outline
- groove - Defines a 3D grooved outline
- ridge - Defines a 3D ridged outline
- inset - Defines a 3D inset outline
- outset - Defines a 3D outset outline
- none - Defines no outline
- hidden - Defines a hidden outline

The following example shows the different outline-style values:



```
A dotted outline.
A dashed outline.
A solid outline.
A double outline.
A groove outline. The effect depends on the outline-color value.
A ridge outline. The effect depends on the outline-color value.
An inset outline. The effect depends on the outline-color value.
An outset outline. The effect depends on the outline-color value.
```

Open the files above (HTML & CSS) in folder supplied.

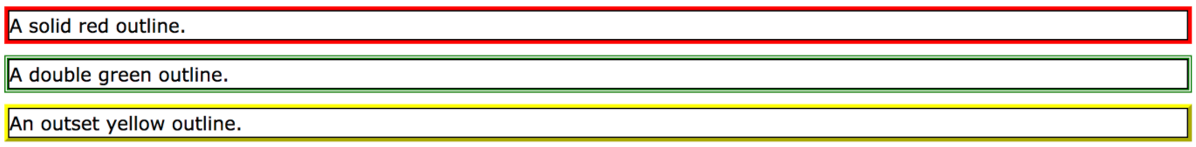
Outline Colour

The outline-colour property is used to set the colour of the outline.

The colour can be set by:

- name - specify a colour name, like "red"
- RGB - specify a RGB value, like "rgb(255,0,0)"
- Hex - specify a hex value, like "#ff0000"
- invert - performs a colour inversion (which ensures that the outline is visible, regardless of colour background)

The following example shows some different outlines with different colours. Also notice that these elements also have a thin black border inside the outline:



A solid red outline.

A double green outline.

An outset yellow outline.

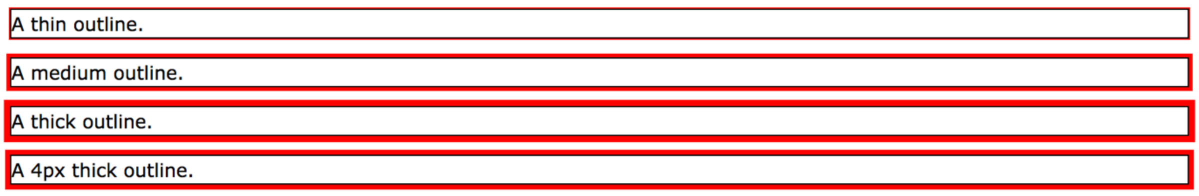
Open the files above (HTML & CSS) in folder supplied.

Outline Width

The outline-width property specifies the width of the outline, and can have one of the following values:

- thin (typically 1px)
- medium (typically 3px)
- thick (typically 5px)
- A specific size (in px, pt, cm, em, etc)

The following example shows some outlines with different widths:



A thin outline.

A medium outline.

A thick outline.

A 4px thick outline.

Open the files above (HTML & CSS) in folder supplied.

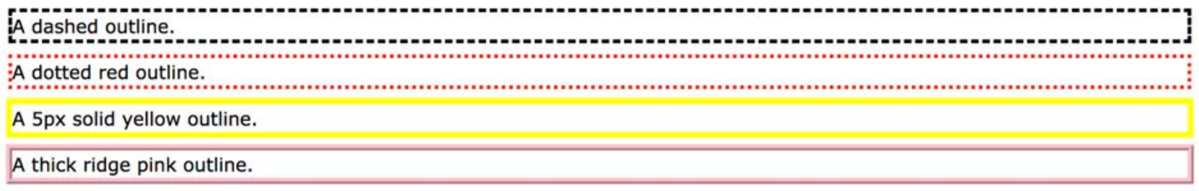
Outline - Shorthand property

The outline property is a shorthand property for setting the following individual outline properties:

- outline-width
- outline-style (required)
- outline-colour

The outline property is specified as one, two, or three values from the list above. The order of the values does not matter.

The following example shows some outlines specified with the shorthand outline property:



A dashed outline.

A dotted red outline.

A 5px solid yellow outline.

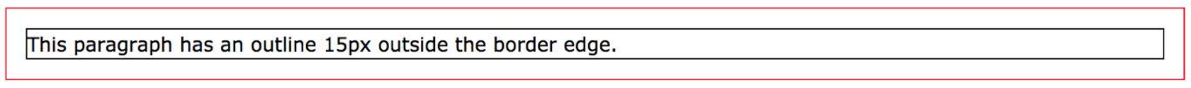
A thick ridge pink outline.

Open the files above (HTML & CSS) in folder supplied.

Outline Offset

The outline-offset property adds space between an outline and the edge/border of an element. The space between an element and its outline is transparent.

The following example specifies an outline 15px outside the border edge:



This paragraph has an outline 15px outside the border edge.

Open the files above (HTML & CSS) in folder supplied.

CSS layout

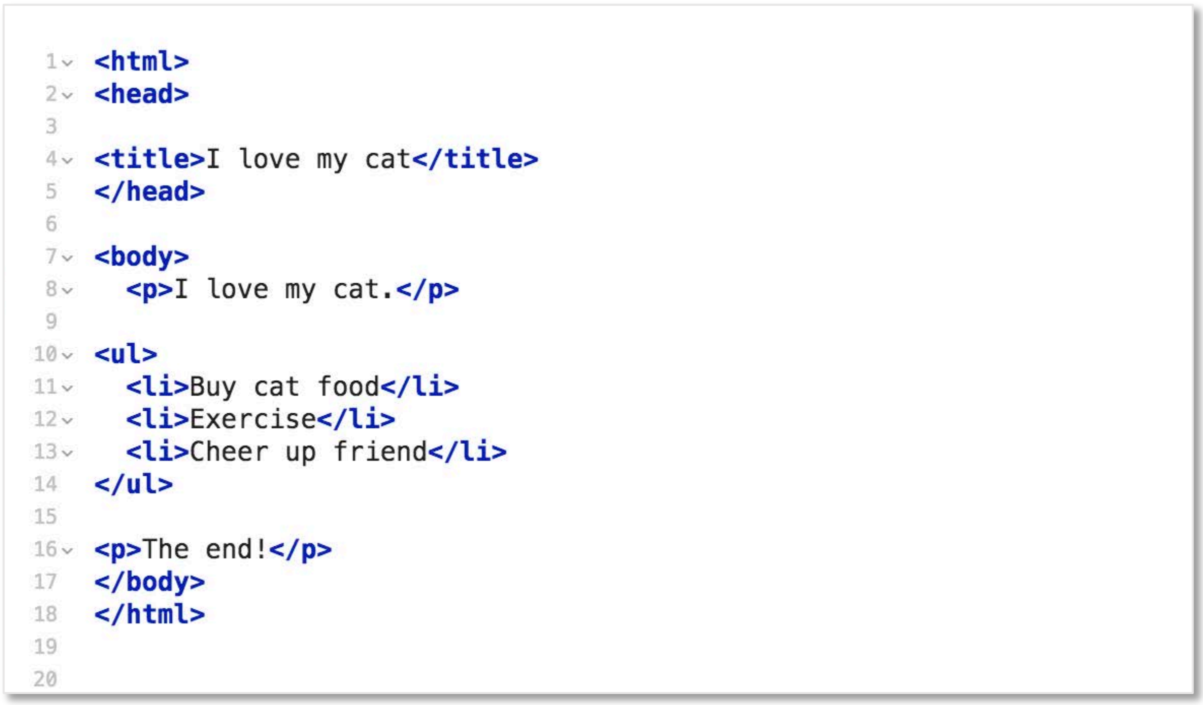
CSS layout overview

CSS page layout techniques allow us to take elements contained in a web page and control where they are positioned relative to their default position in normal layout flow, the other elements around them, their parent container, or the main viewport/window.

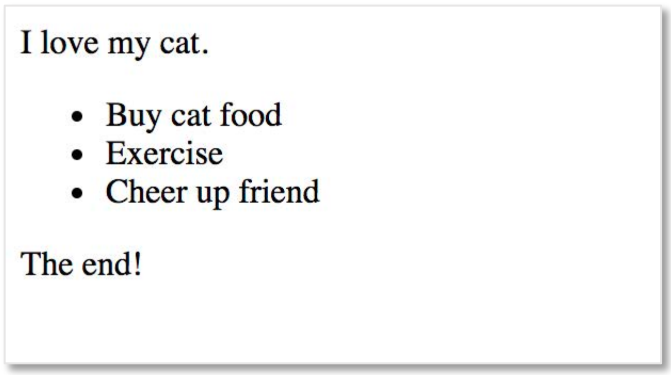
Each technique has its uses, advantages, and disadvantages, and no technique is designed to be used in isolation. By understanding what each method is designed for you will be in a good place to understand which is the best layout tool for each task.

Normal Flow

Normal flow is how the browser lays out HTML pages by default when you do nothing to control page layout.



```
<html>
<head>

<title>I love my cat</title>
</head>

<body>
  <p>I love my cat.</p>

<ul>
  <li>Buy cat food</li>
  <li>Exercise</li>
  <li>Cheer up friend</li>
</ul>

<p>The end!</p>
</body>
</html>
```



I love my cat.

- Buy cat food
- Exercise
- Cheer up friend

The end!

Note here how the HTML is displayed in the same order in which it appears in the source code, with elements stacked up on top of one another — the first paragraph, followed by the unordered list, followed by the second paragraph.

The elements that appear one below the other are described as block elements, in contrast to inline elements, which appear one beside the other, like the individual words in a paragraph.

When you use CSS to create a layout, you are moving the elements away from the normal flow, but for many of the elements on your page the normal flow will create exactly the layout you need. This is why starting with a well-structured HTML document is so important, as you can then work with the way things are laid out by default rather than fighting against it.

The methods that can change how elements are laid out in CSS are as follows:

- ***The display property*** - Standard values such as block, inline or inline-block can change how elements behave in normal flow. We then have entire layout methods that are switched on via a value of display, for example CSS Grid and Flexbox.

- ***Floats*** - Applying a float value such as left can cause block level elements to wrap alongside one side of an element, like the way images sometimes have text floating around them in magazine layouts.

- ***The position property*** - Allows you to precisely control the placement of boxes inside other boxes. static positioning is the default in normal flow, but you can cause elements to be laid out differently using other values.

- ***Table layout*** - features designed for styling the parts of an HTML table can be used on non-table elements using display: table and associated properties.

- ***Multi-column layout*** - The multi-column layout properties can cause the content of a block to layout in columns, as you might see in a newspaper.

The display property

This property allows us to change the default way something displays. Everything in normal flow has a value of display, used as the default way that elements they are set on behave. For example, the fact that paragraphs in English display one below the other is due to the fact that they are styled with display: block. If you create a link around some text inside a paragraph, that link remains inline with the rest of the text, and doesn't break onto a new line. This is because the <a> element is display: inline by default.

You can change this default display behaviour. For example, the <li> element is display: block by default, meaning that list items display one below the other in our English document. If we change the display value to inline they now display next to each other, as words would do in a sentence. The fact that you can change the value of display for any element means that you can pick HTML elements for their semantic meaning, without being concerned about how they will look. The way they look is something that you can change.
In addition to being able to change the default presentation by turning an item
from block to inline and vice versa, there are some bigger layout methods that start out as a value of display. However when using these you will generally need to invoke additional properties. The two values most important for our purposes when discussing layout
are display: flex and display: grid.

Flexbox

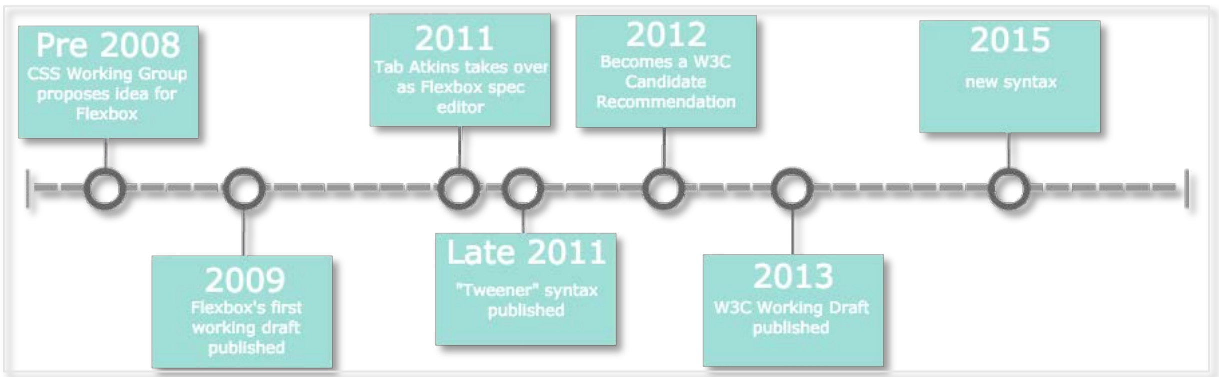
Flexbox is the short name for the Flexible Box Layout Module, designed to make it easy for us to lay things out in one dimension - either as a row or as a column. To use flexbox, you apply display: flex to the parent element of the elements you want to lay out; all its direct children then become flex items.

The HTML markup in the example below gives us a containing element, with a class of wrapper, inside which are three <div> elements. By default these would display as block elements, below one another, in our English language document.

However, if we add display: flex to the parent, the three items now arrange themselves into columns. This is due to them becoming *flex items* and using some initial values that flexbox gives them. They are displayed as a row, because the initial value of flex-direction is row. They all appear to stretch to the height of the tallest item, because the initial value of the align-items property is stretch. This means that the items stretch to the height of the flex container, which in this case is defined by the tallest item. The items all line up at the start of the container, leaving any extra space at the end of the row.

Open the Flexbox sample files above (HTML & CSS) in folder supplied.

In addition to the above properties that can be applied to the flex container, there are properties that can be applied to the flex items. These properties, among other things, can change the way that the items flex, enabling them to expand and contract to fit into the available space.


Pre 2008
CSS Working Group proposes idea for Flexbox
2009
Flexbox's first working draft published
2011
Tab Atkins takes over as Flexbox spec editor
Late 2011
"Tweener" syntax published
2012
Becomes a W3C Candidate Recommendation
2013
W3C Working Draft published
2015
new syntax

Grid Layout

While flexbox is designed for one-dimensional layout, Grid Layout is designed for two dimensions — lining things up in rows and columns.

Once again, you can switch on Grid Layout with a specific value of display — display: grid. The below example uses similar markup to the flex example, with a container and some child elements. In addition to using display: grid, we are also defining some row and column tracks on the parent using the grid-template-rows and grid-template-columns properties respectively. We've defined three columns each of 1fr and two rows of 100px. I don't need to put any rules on the child elements; they are automatically placed into the cells our grid has created.

Open the Grid 1 example above (HTML & CSS) in folder supplied.

Once you have a grid, you can explicitly place your items on it, rather than relying on the auto-placement behaviour seen above. In the second example below we have defined the same grid, but this time with three child items. We've set the start and end line of each item using the grid-column and grid-row properties. This causes the items to span multiple tracks.

Open the Grid 2 example above (HTML & CSS) in folder supplied.

There are other layout methods, which are less important for the main layout structures of your page but can still help you achieve specific tasks. These include floats and positioning.

TEACHER TIP
Use the GridbyExample website to learn more about using grids with CSS - https://gridbyexample.com/examples/page-layout/#layout1

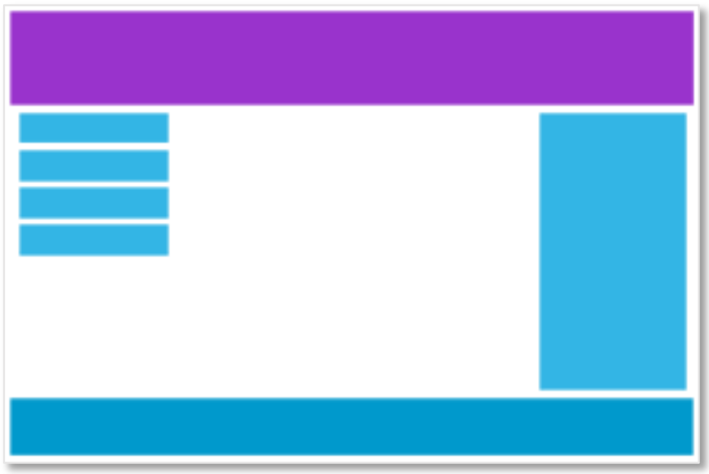
Responsive Web Design

Introduction

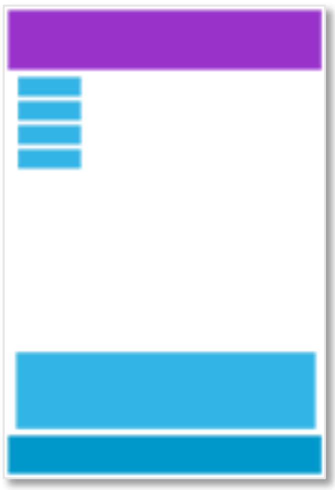
- Responsive web design makes your web page look good on all devices.
- Responsive web design uses only HTML and CSS.
- Responsive web design is not a program or a JavaScript.

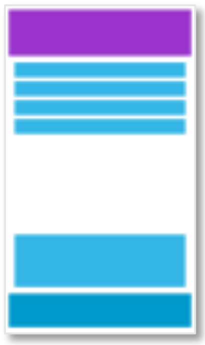
The Best Experience For All Users

Websites can be viewed using many different devices: desktops, tablets, and phones. Your web page should look good, and be easy to use, regardless of the device.
Web pages should not leave out information to fit smaller devices, but rather adapt its content to fit any device:



Desktop


Tablet


Phone

It is called responsive web design when you use CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen.

TEACHER TIP
To learn more on Responsive Web Design including details on Viewpoint, Grid View, Media Queries, Image, Videos, Frameworks and to try some templates out please see https://www.w3schools.com/css/css_rwd_viewport.asp