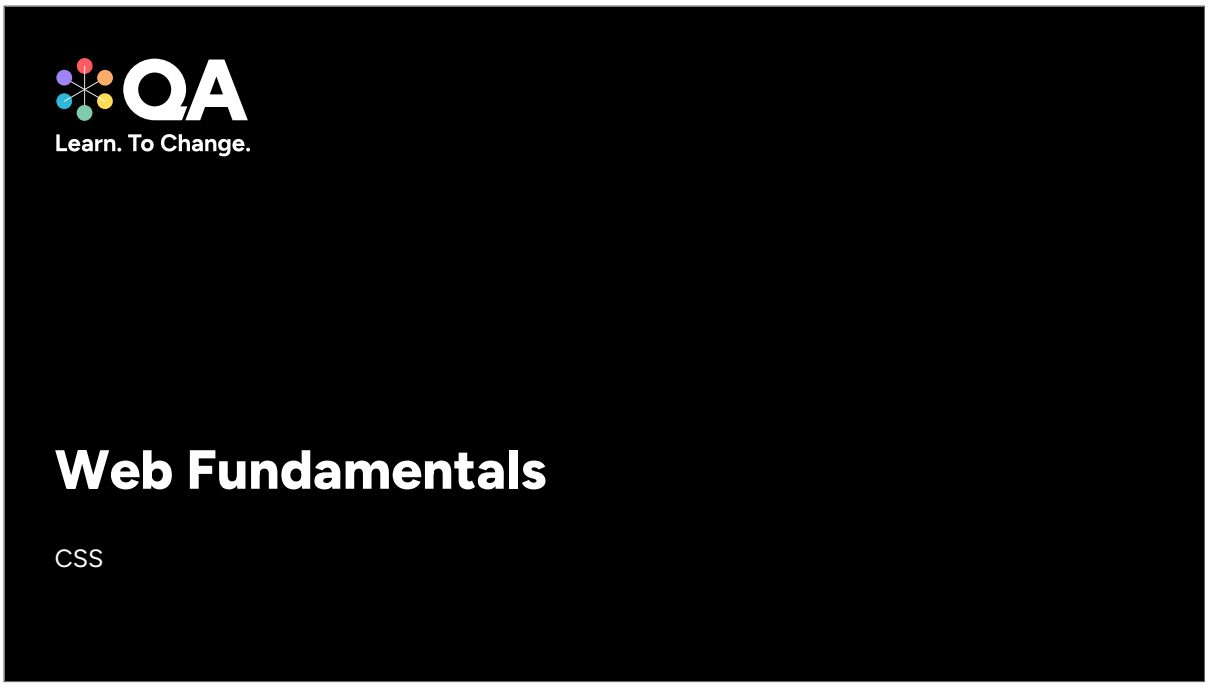

QA
Learn. To Change.
Web Fundamentals
CSS

QA

Learning Objectives

- Understand what CSS is
- Understand how CSS can be applied to web pages
- Understand the syntax of writing CSS rules
- Be able to select elements to apply CSS to
- Be able to work with Text, Colours and Images
- Be able to work with the Box Model and position elements
- Be able to style lists and tables
- Be able to add CSS animations, transforms and transitions to elements

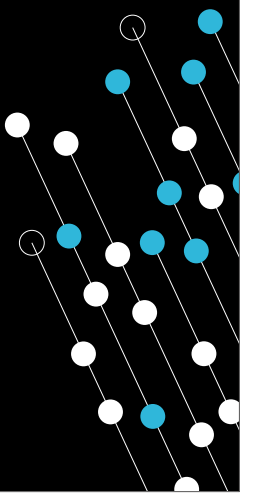
QA

Cascading Style Sheets

- CSS application and syntax
- CSS and the DOM
- Inheritance and Selecting Elements
- Text and Colours
- Measurement Units
- Images and Backgrounds
- The Box Model
- Lists and Tables
- Animations, Transitions and Transformations

CSS Application and Syntax

CSS Fundamentals



QA

Cascading Style Sheets

- **CSS stands for Cascading Style Sheets**
- **It describes how HTML elements are to be displayed**
  - This could be on the screen, on other media or even how it should be printed on paper
- **Can control the layout of multiple web pages from a set of rules**
- **Styling can be applied in one of four ways:**
  - Inline – defined in the actual element to style
  - In an embedded stylesheet on the page – defined on a per-page basis
  - In an external style sheet linked to the page – defined inside a separate .css file
  - By linking in some existing CSS (almost never to be used)

HTML was never intended to provide details about how to format the content – just describe what the content is.

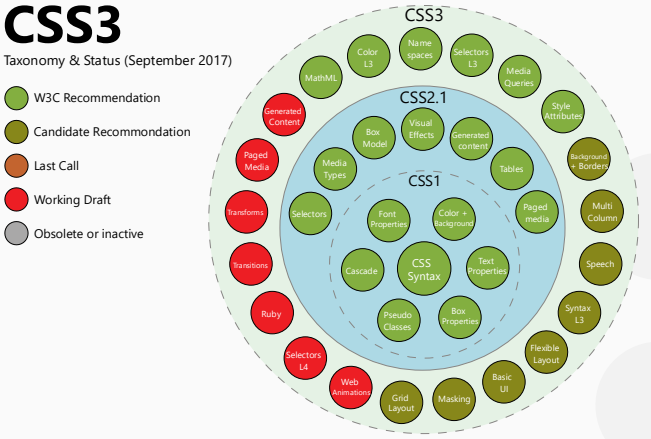
HTML 3.2 introduced some styling tags like <font> and these were used all over the markup to help show how to display it.

CSS was introduced by W3C to help keep content and styling separate.

QA

CSS Taxonomy And Status (2017)

- This is a popular diagram to show the history of CSS progression

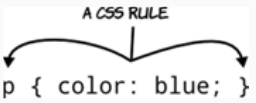


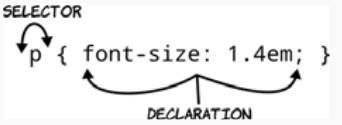
https://commons.wikimedia.org/wiki/File:CSS3_taxonomy_and_status_by_Sergey_Mavrody.svg

QA

CSS Basic Syntax

- Rules, selectors, properties, and values
- A CSS style sheet is made up of rules
- Here are three examples CSS rules:





A CSS rule is made up of a selector and a semicolon-separated list of declarations inside brackets. Each declaration has a property and a value separated by a colon. If an element in an associated HTML document matches a selector in the style sheet, then the declarations will be applied to that element.

QA

Inline Styles

- **style** attribute can be used on any HTML tag
- Affects that HTML tag only

```
<p style="margin-left: 1in; margin-right: 1in; line-height:200%">
  This text will be shown with one-inch left and right margins, and
  double-spaced.
</p>
<p>
  This text is formatted as normal for &lt;p&gt; tags.
</p>
```

Style information can be embedded within HTML tags using the **STYLE** attribute. Style information held in this way is not global to a document but local to that particular tag.

For example:

```
<p style="margin-left: 0.5in; margin-right: 0.5in">
This text will be shown with half-inch left and right margins.
</p>
<p>
This text is formatted as normal for &lt;p&gt; tags.
</p>
```

QA

Embedded Style Sheets

Use **<style> … </style>** inside the **<head>** tag

A style sheet definition contains a list of

- HTML tags, and
- Associated format information for that tag

```
.. .. ..
<style>
  h1 { font-size: 15pt; font-weight:bold}
  p { font: bold italic 12pt/20pt times, serif}
</style>
.. .. ..
```

Style sheets can be embedded with the HTML document, between **<style>** and **</style>** container tags, which in turn must be placed between the **<html>** and **<body>** tags of the page.

The **<style>** tag has a **TYPE** attribute which is **text/css** for cascading style sheets.

The style sheet definition consists of a list of HTML tags (style sheets may be applied to any HTML tag) and formatting instructions. Here a simple example which assigns formatting to the **<h1>** and **<p>** styles.

```
<style type="text/css">
h1 {font-size:15pt;font-weight:bold}
p {font:bolditalic 12pt/20pt Times,serif}
</style>
```

QA

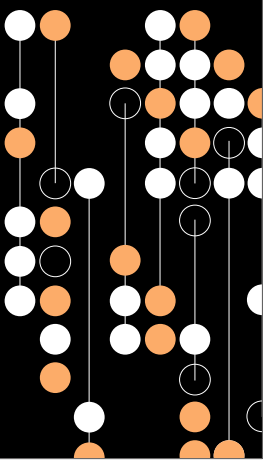
External Style Sheets

- Put all CSS in separate file and then link to it from each page
  - In same format that it appeared in the Internal Style Sheets
- **`<link>`** element references an external style sheet
- Should appear in the head of the document

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

CSS and the DOM

CSS Fundamentals



QA

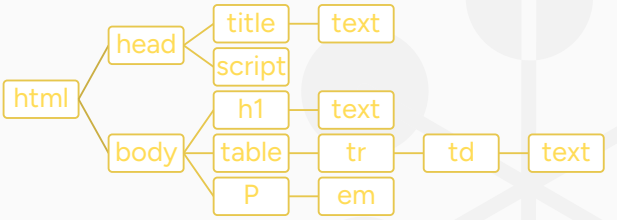
Recall: the Document Object Model

HTML documents have a hierarchical structure that form the DOM

- Every element, except <html> is contained within another
- Creating a parent/child relationship

A DOM tree contains two types of elements

- Nodes.
- Text.



HTML documents have what is called a hierarchical structure. Each element (or tag) except the top <html> tag is contained in another element, its parent. This element can in turn contain child elements. You can visualise this as a kind of organisational chart or family tree.

HTML has valid rules for how this DOM fits together and renders the markup delivered from the server into a client side DOM. For all intents and purposes, the DOM is a complex array.

QA

HTML markup to DOM object (1)

Consider the following HTML

```
<img id="myImage" src="image.gif" alt="An image" title="This is an image"/>
```

The tag has a type of **<img>** and four attributes

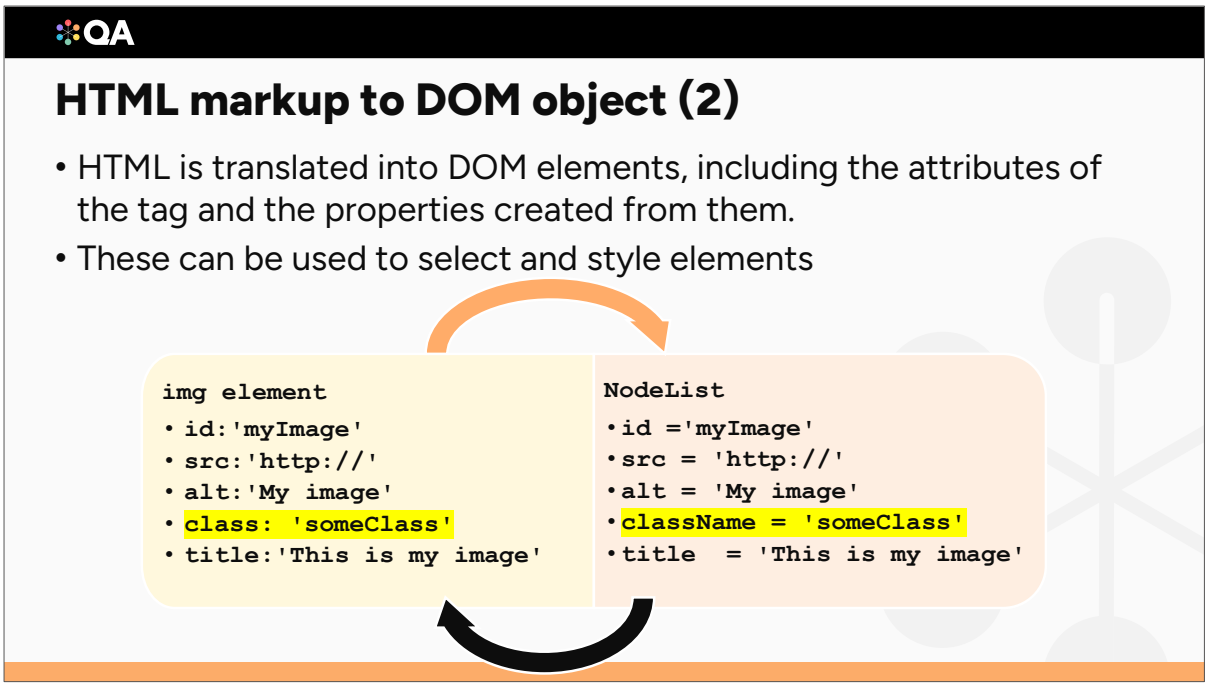
- **id**
- **src**
- **alt**
- **title**

The element is read and interpreted by the browser into a DOM

- Each element becomes a NodeList object
- Assigned a property based on the html attribute

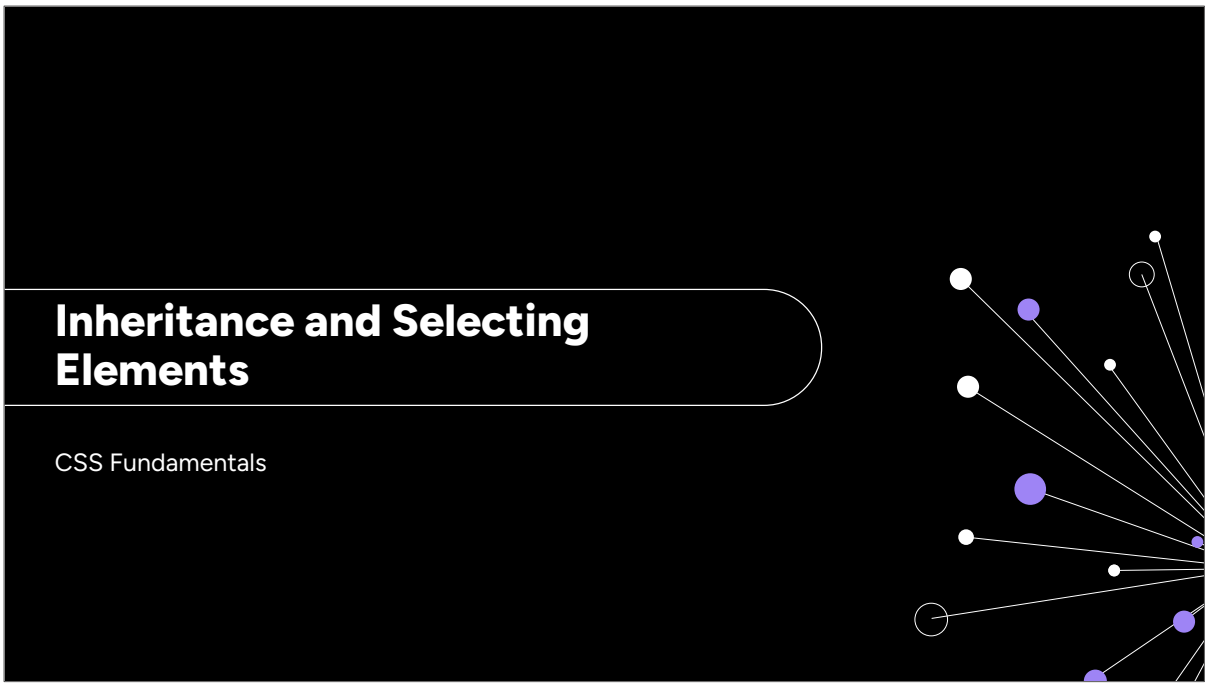
HTML gives the browser instructions on how to render the order and set the properties of the objects. Each element in HTML can have zero or more attributes assigned to it. So, in the example above each element has properties and attributes assigned to it.

These properties and attributes are initially assigned to the DOM elements as a result of parsing their HTML markup and can be changed dynamically under script control. To make sure we have our terminology and concepts straight, consider the following HTML markup for an image element on the slide above.



For the most part, the name of a JavaScript attribute property matches that of any corresponding attribute, but there are some cases where they differ. For example, the class attribute in this example is represented by the className attribute property.

Every element mapped into the DOM is, in fact, an object literal of key value pairs as we discussed in the previous chapter. If we can select the element we want from the DOM, we can manipulate it in exactly the same way.


Inheritance and Selecting Elements
CSS Fundamentals

QA

Selecting Elements To Style

- Select all tags of a particular type
- Select tags dependent on the relationship to others in the DOM
- Select tags based on their id or class attribute
- Select tags based on other attributes
- Select tags based on a combination of the above

QA

Selecting all tags of a particular type

This is as simple as creating a rule for the tag name and nothing else

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

- ***Assuming that this is the only CSS applied to the page (more on that later…)***

```
div { background-color: red }
```

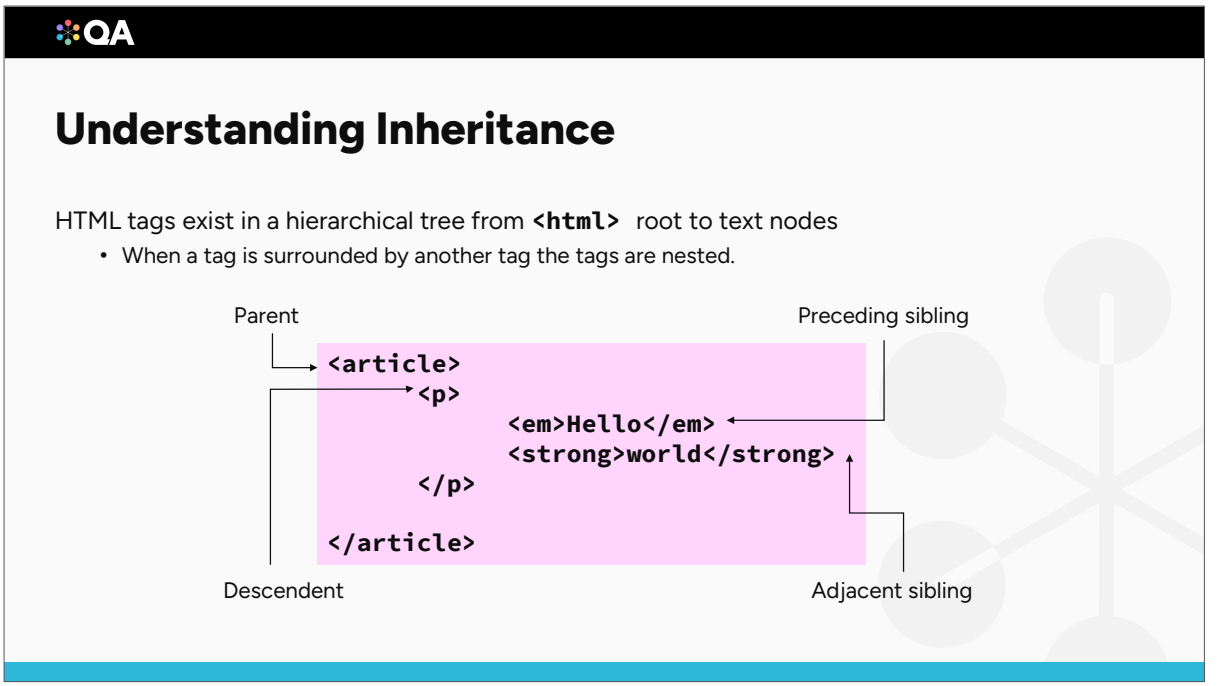
- Would make all text in *any* `p` element **green**

```
h1, a {color: pink }
```

- Would make the background colour of *any* `div` element **red**

- Multiple elements can be selected by putting a comma between them

Note the spelling of color – it's not a typo, CSS uses American spellings!



When a tag is surrounded by another tag—one inside another—the tags are nested. In a nested set, the outer element is called the parent, and the inner element the child. The child tag and any children of that child tag are the parents' descendants.

HTML tags exist in a hierarchical tree from <html> root to text nodes

When a tag is surrounded by another tag the tags are nested.

- Tags in the same parent are called siblings
- Tags immediately next to each other are adjacent siblings
- Parent elements contain other elements (children)
- Child elements will often inherit styles from a parent element.
- Descendent elements are any elements within another element.

The article element is the parent to the elements created by the paragraph, strong, and emphasis tags, which are its descendants. Only the paragraph tag is a direct child.

QA

Hierarchical Inheritance

Elements inherit from containing parents

- So we only need to define a style rule at the highest level
- We can then override rules at descendent levels

Complex hierarchies are difficult to manage

- Chrome's developer tools help greatly
- Showing which styles are applied
- Where they come from
- If rules are being overridden
- The order in which they're applied

A selector's specificity is calculated as follows:

count the number of ID selectors in the selector (= a)

count the number of class selectors, attributes

selectors, and pseudo-classes in the selector (= b)

count the number of type selectors and pseudo-elements in the selector (= c)

ignore the universal selector

Selectors inside the negation pseudo-class (:not) are counted like any other, but the negation itself does not count as a pseudo-class

Concatenating the three numbers a-b-c gives the specificity (except 1 a is always more specific than any number of b or c and likewise 1 b more than any number of c)

QA

Select tags dependent on the relationship to others in the DOM

Descendant selector – put a space between the parent and child – all descendants will be styled

```
ul li { color: purple }
```

- Would make all text in *any* `li` that is a *descendant* of a `ul` **purple**

Child selector – put a > between the parent and child – any direct child will be styled

```
section > p { color: brown }
```

- Would make all text in *any* `p` that is a *direct descendant* of a `section` **brown**

Adjacent selector – put a **+** between the siblings – last sibling listed will be styled

```
h2 + p { color: black }
```

- Would make all text in any `p` that *immediately follows* a `h2` element will be **black**

Sibling selector – change the + for a ~ to select *any following element*

X Y - Descendant Selector

The descendant selector matches all elements that are descendants of a specified element. The first simple selector within this selector represents the *ancestor element*—a structurally superior element, such as a parent element, or the parent of a parent element, and so on

X>Y - Child Selector

This selector matches all li elements that are the immediate children of a ul element—that is, all li elements that have a ul element as a parent. This is referred to as a child selector.

X+Y - Adjacent Sibling Selector

The adjacent sibling selector selects the element that is the adjacent siblings of a specified element. Sibling elements must have the same parent element, and "adjacent" means "immediately following," so there can be no elements between the sibling elements. The combinator in an adjacent sibling selector is a plus character.

X~Y - Sibling Selector

The sibling selector matches all elements that are siblings of a given element. The elements don't have to be adjacent siblings, but they have to have the same parent. The sibling combinator is similar to X + Y, however, it's less strict. While an adjacent selector (ul + p) will only select the first element that is immediately preceded by the former selector, this one is more generalised.

QA

Select tags based on their id or class attribute

id selector – put a # before the name of the id to be styled

```
#myChosenId { color: purple }
```

- Would make all text in *any* **element** that has an `id` attribute of `myChosenId` purple
- **Note:** an **id** should be *unique* within a page, if *more than one* is needed a **class** should be used

class selector – put a `.` Before the name of the class to be styled

```
.myChosenClass { color: brown }
```

- Would make all text in *any* **element** that has a `class` attribute of of a `myChosenClass` brown

```
<p id="myChosenId">A paragraph with myChosenId</p>
<p class="myChosenClass">A paragraph with myChosenClass</p>
```

QA

Selecting sets of elements with pseudo-classes

Selecting first and last element

```
ul li:first-child { background-color: red; }
ul li:last-child { background-color: red; }
```

Selecting an element by its ordering

```
li:nth-child(3), li:nth-child(5) { background-color: red; }

li:nth-child(2n + 1) { background-color: red; }

li:nth-child(odd) { background-color: blue; }

li:nth-child(even) { background-color: green; }
```

QA

Selecting sets of elements with pseudo classes

More selection patterns

```
ul:last-child { background-color: red; }
```

```
ul:first-child:last-child { background-color: red; }
```

Selecting by types of element

```
article:first-of-type { background-color: red; }
```

```
p:last-of-type { background-color: red; }
```

Choosing what isn't

```
input:not([type=checkbox]):not([type=radio]) {
    display: block; width: 12em;
}
```

ul:last-child { background-color: red; } Without specifying that only <li> elements that were last children should be styled, the rule now selects any element that is a last child.

ul :first-child:last-child { background-color: red; } This rule selects elements that are both a first and a last child and are descendants of the <ul> element.

article:first-of-type { background-color: red; } This selector will always apply to the first article element on the page, as well as any other first article elements further down the tree.

p:last-of-type { background-color: red; } The last-of-type pseudo-class works in the same way, except in reverse.

CSS3 also gives us the ability to select elements that aren't the first child or don't have particular attributes, with the :not pseudo-class.

QA

Pseudo classes and elements

(Dynamic) pseudo classes, elements...

- Applying to user actions (pseudo classes)

```
:active { color: blue; }
:hover { color: blue; }
:focus { color: blue; }
:link { color: blue; }
:visited { color: blue; }
```

- Applying to placement (pseudo elements)

```
::after { color: blue; }
::before { color: blue; }
::first-letter { color: blue; }
::first-line { color: blue; }
```

- Selection pseudo element

```
::selection { color: blue; }
```

In CSS 2.1, style is normally attached to an element based on its position in the document tree. This simple model is sufficient for many cases, but some common publishing scenarios may not be possible due to the structure of the document tree. For instance, in HTML 4 no element refers to the first line of a paragraph, and therefore no simple CSS selector may refer to it.

CSS introduces the concepts of pseudo-elements and pseudo-classes to permit formatting based on information that lies outside the document tree.

Pseudo-elements create abstractions about the document tree beyond those specified by the document language. For instance, document languages do not offer mechanisms to access the first letter or first line of an element's content. CSS pseudo-elements allow style sheet designers to refer to this otherwise inaccessible information.

Pseudo-classes classify elements on characteristics other than their name, attributes or content; in principle characteristics that cannot be deduced from the document tree. Pseudo-classes may be dynamic, in the sense that an element may acquire or lose a pseudo-class while a user interacts with the document.

QA

Pseudo classes specificity

Recall: in the cascade styles are sorted by specificity

- Latter rules are more specific that earlier rules

```
a               {color: black; }
a:link          {color: blue; }
a:visited       {color: red; }
a:hover         {color: green; }
a:active        {color: orange; }
```

Note that the A:hover must be placed after the A:link and A:visited rules, since otherwise the cascading rules will hide the 'color' property of the A:hover rule.

Similarly, because A:active is placed after A:hover, the active colour (orange) will apply when the user both activates and hovers over the A element.

QA

before: and after:

Used to insert content before or after an element

- Can be specific content, counters or values of attributes

Specify style and content of inserted content

- content: normal | none | <string> | <uri> | <counter> | attr(<identifier>) | open-quote | close-quote | no-open-quote | no-close-quote | inherit

```
p.note:before { font-weight: bold; content: "Note: "; }

h1:before {
  content: "Chapter " counter(chapter) ".";
  counter-increment: chapter;
}
```

The 'content' property is used with the :before and :after pseudo-elements to generate content in a document. Values have the following meanings:

none The pseudo-element is not generated.

normal Computes to 'none' for the :before and :after pseudo-elements.

<string> Text content.

<uri> The value is a URI that designates an external resource (such as an image).

<counter> Counters may be specified with two different functions: 'counter()' or 'counters()'. The former has two forms: 'counter(name)' or 'counter(name, style)'. The latter function also has two forms: 'counters(name, string)' or 'counters(name, string, style)'.

open-quote and close-quote These values are replaced by the appropriate string from the 'quotes' property.

no-open-quote and no-close-quote Introduces no content, but increments (decrements) the level of nesting for quotes.

Here is a style sheet that numbers paragraphs (p) for each chapter (h1). The paragraphs are numbered with roman numerals, followed by a period and a space:

```
p {counter-increment: par-num}

h1 {counter-reset: par-num}

p:before {content: counter(par-num, upper-roman) ". "}
```

QA

Choosing elements by their attribute

- = operator finds attributes whose value exactly matches

```
a[href="http://www.qa.com"] { color: blue; }
```

- ^= operator finds attributes starting with a value

```
a[href^="http:"] { color: blue; }
```

- $= operator finds any element attributes ending with a value

```
[src$=".png"] { color: red; }
```

- *= operator finds attributes containing the value

```
[id*="stuff"] { color: red; }
```

CSS3 makes it possible to write selectors that target elements based on the values in their attribute. These are called *attribute selectors*.

Note that the attribute itself is a selector in its own right.

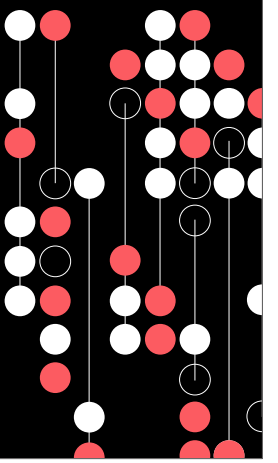
QA

Quick Lab 6 – CSS Selectors

Apply selectors to style rules to apply styling to particular elements

Text and Colours

CSS Fundamentals



QA

Working with fonts – setting the character type

As previously noted it is important to set the encoding type of a document.

- In HTML5

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

- In XHTML/HTML4

```
<meta http-equiv="Content- Type" content="text/html; charset="utf-8" />
```

A character set is a list of character codes your browser will accept

- If it does not understand a character a glyph will appear in its place
- You are also leaving yourself exposed to dangerous JavaScript atta

QA

Font Families

CSS Defines five font families to which most fonts are categorised

- Serif - Times New Roman
- Sans-serif - Arial
- Monospace - Courier New
- Cursive - Brush Script
- Fantasy - Papyrus

```
font-family: Helvetica, Verdana, Arial, sans-serif
```

There are also dingbats and other symbol library fonts

Plus HTML character entities

- `&pound;` for a GBP symbol as an example

Fonts are set in a comma delimited list

- Browser checks if font is available, used if present moves on if not

CSS defines five generic font families that categorises most fonts.

- Serif fonts and letters the end with a ornamental tail and can improve legibility in print-based media. They are best suited for display of a larger text on screen or print stylesheets.
- Sans serif fonts do not have the distinctive serif tail they work well on the screen especially for smaller text.
- Monospace fonts categorised by symbols that always take the same amount space with no letter overlap or kerning. Very useful for showing technical information such as code.
- Cursive fonts mimic calligraphic hand written styles and are best used for large headings else readability is reduced.
- Fantasy Art decorative fonts that don't fit into any of the other categories they tend to be heavily stylised or ornamental to be used in the same way as cursive fonts.

There are also symbol library fonts including dingbats. These are very useful as a providing icons and pictograms but how much or lower download footprint than a similar image file.

Font information is normally set as a family by creating a comma delimited list of fonts the browser uses the first font available on the local system. It is common to use one of the CSS high-level categories as your final font choice allowing a browser to use the systems default sans serif font. If no match is found the browser uses its preferred default font which would be Times New Roman on most Windows PCs

QA

Other Font Settings

- Additional typography properties can be set:

Property	Usage
`font-size`	Font size can use any of the units previously discussed or a value between xx-small and xx-large
`font-weight`	font-weight controls the normal weight of the font normal | bold or a weight scale between 100 and 900
`font-style`	Normal, italic or oblique – if no oblique is present italic will be used.
`line-height`	The height of each line of text known as leading
`vertical-align`	Sets the alignment of the text in relation to the line box.

QA

Setting fonts as a compound rule

Fonts need to be set in a very specific way using CSS.

- Requiring a minimum set of keywords and a specific order
- The most basic rule requires:

```
font: <font-size> <font-family>;
```

When using a complex rule optional values precede the mandatory

```
font: italic small-caps 1.2em Georgia, serif;
```

```
font: 100%/2.5 Helvetica,
```

With the exception of a sneakily inserted line-height

- Note the lack of measurement unit
- You can add them but it can cause issues

We've dealt with compound rules already and most do not require rules added in a specific order. Fonts are an exception to this. Font requires a minimum set of keywords and a specific order. If you reverse them, or leave one out, and any modern browser will just ignore the declaration outright. When the rule becomes complex with one exception other values must come before the required values.

So we can see that the rules for compound font tags are a little odd, and they get odder! Line height defines how much space should exist between each line of text in our code. We can set this separately or wrap it up in the font size settings. There's no space between the font-size and line-height values, just a forward slash. Adding the line-height to a font declaration is always optional, but if you do include it, its placement is not. You must immediately follow the font's size with a forward slash and the line-height value.

You will note the line height is without a unit type. You could give a unit but it can have unexpected knock on effects to descendent elements. For example if you set the following CSS:

```
ul {font-size: 15px; line-height: 1em;}
li {font-size: 10px;} small {font-size: 80%;}
<ul>
    <li>I'm a list item with <small>small text</small>.</li>
</ul>
```

The <li>, using % based units now has to factor in em units. The result may be chaotic! If we omit the unit it acts as a multiple of the elements unit.

QA

Text Alignment and Other Properties

Property	Description	Common Values
`color`	Sets the text colour for this and child elements	`Any valid colour`
`text-align`	Sets the horizontal alignment of text	`left, right, center, justify`
`text-decoration`	Sets or removes decorations from text	`none`
`text-transform`	Specifies case for text	`uppercase, lowercase, capitalize`
`text-indent`	Specifies indentation of first line of text	`Any valid measurement`
`letter-spacing`	Specifies space between characters in text	`Any valid measurement`
`line-height`	Sets space between lines	`Any valid measurement`
`text-direction`	Changes the direction of text	`rtl, ltr`
`word-spacing`	Sets space between words	`Any valid measurement`
`text-shadow`	Adds shadow to text – gives horizontal, vertical and colour of shadow	`3px 3px green`

QA

Adding a drop shadow

Drop shadow is back as if the 1990's never happened!

```
.shadow {
      text-shadow: 10px 8px 20px rgb(56, 52, 153);
}
```

text-shadow requires the following properties:

- X, Y offset
- Amount of blur
- Colour
- Corresponding box shadow rule

```
.shadow {
      box-shadow: 3px 3px 3px 3px rgb(0, 0, 119);
}
```

Text-shadow is used for text elements not structural one e.g. <aside>. There is a corresponding box shadow for block level objects

Shadows may inherit, it is possible to set a text-shadow rule to none to override a previous set shadow. They also work very well with the hover pseudo class.

box-shadow property can accept a comma-separated list of shadows, each defined by 2-4 length values (specifying in order the horizontal offset, vertical offset, optional blur distance and optional spread distance of the shadow), an optional colour value and an optional 'inset' keyword (to create an inner shadow, rather than the default outer shadow) e.g.:

box-shadow: inset -5px -5px 5px #888;

The box-shadow property allows elements to have multiple shadows, specified by a comma separated list. When more than one shadow is specified, the shadows are layered front to back, as in the following example.

box-shadow: 0 0 10px 5px black, 40px -30px lime, 40px 30px 50px red, -40px 30px yellow, -40px -30px 50px blue;

QA

Color values and format

W3C specifies 4 numerical colour value methods:

- RGB, RGBA, HSL and HSLA

There are also 16 basic named colour values that can be used in CS

- Additional 128 colours are named in the extended set
  https://www.w3.org/TR/2018/REC-css-color-3-20180619/#svg-co

Color names and sRGB values

Named	Numeric	Color name	Hex rgb	Decimal
		black	#000000	0,0,0
		silver	#C0C0C0	192,192,192
		gray	#808080	128,128,128
		white	#FFFFFF	255,255,255
		maroon	#800000	128,0,0
		red	#FF0000	255,0,0
		purple	#800080	128,0,128
		fuchsia	#FF00FF	255,0,255
		green	#008000	0,128,0
		lime	#00FF00	0,255,0
		olive	#808000	128,128,0
		yellow	#FFFF00	255,255,0
		navy	#000080	0,0,128
		blue	#0000FF	0,0,255
		teal	#008080	0,128,128
		aqua	#00FFFF	0,255,255

QA

Color values and format RGB/RGBA

Used to specify RED, GREEN and BLUE values

- Can be done with Hexadecimal or as a set of 3 numeric values (either integer or percentage)

```
em { color: blue; }                 /* #rgb */
em { color: #ff0000; }              /* #rrggbb */
em { color: rgb(255,0,0); }
em { color: rgb(100%, 0%, 0%); }
```

The A value can be used to represent ALPHA for opacity of the colour

- Cannot be used with HEX values

```
em { color: rgb(255,0,0); }       /* integer range 0 - 255 */
em { color: rgba(255,0,0,1); }    /* the same, with explicit opacity of 1 */
em { color: rgb(100%,0%,0%); }    /* float range 0.0% - 100.0% */
em { color: rgba(100%,0%,0%,1); } /* the same, with explicit opacity of 1 */
```

The W3C recommendation states:

Note. If RGBA values are not supported by a user agent, they should be treated like unrecognized values per the CSS forward compatibility parsing rules ([CSS21], Chapter 4). RGBA values must *not* be treated as simply an RGB value with the opacity ignored.

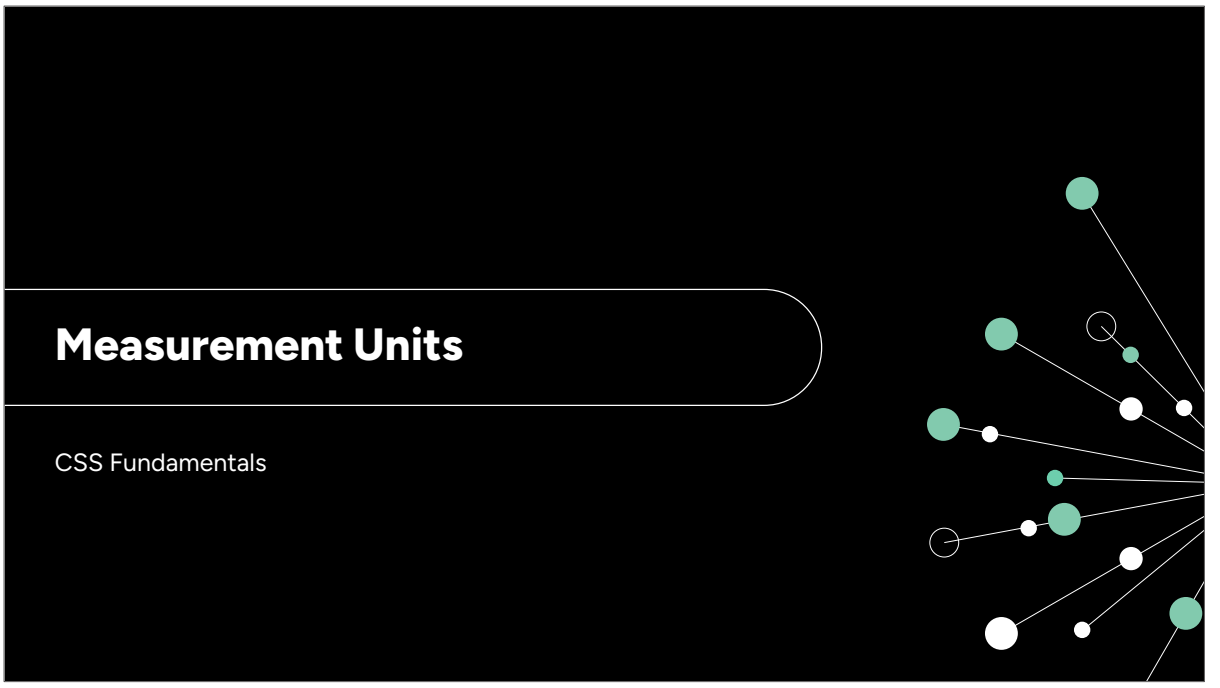
https://www.w3.org/TR/2018/REC-css-color-3-20180619/#rgb-color

QA

Color values and format – HSL/HSLA

- RGB is hardware oriented and harps back to the days when CRT were used in monitors
  - Hue is represented as an angle of the colour circle – measured in degrees and value is used in CSS
- HSL are encoded as Hue, Saturation and Lightness
  - Saturation and Lightness are represented as percentages
    - 100% is full saturation and 0% is a shade of grey
    - 0% lightness is black, 100% is white and 50% is 'normal'

```
* { color: hsl(0, 100%, 50%); }          /* red */
* { color: hsl(120, 100%, 50%); }        /* lime */
* { color: hsl(120, 100%, 25%); }        /* dark green */
* { color: hsl(120, 100%, 75%); }        /* light green */
* { color: hsl(120, 75%, 75%); }         /* pastel green, and so on */
```


Measurement Units
CSS Fundamentals

QA

Element Sizing

Sizing elements can be achieved in a number of different ways:

Pixels (px) - a fixed measurement based on the size of a pixel

```
img { width: 150px; }
```

Ems (em) - a relative unit that equates to the font size of the element.

- An em unit is relative to the parent element's font size.

```
article{ width: 3em; }
```

Points (pts) - Points are an absolute unit equal to 1/72 of an inch

- Points can be useful when setting type sizes for print

```
body{ font-size: 12pt; }
```

% - Size is relative to the containing element

```
p{ width: 50%; }
```

Dimensions and other measurements, such as font-size, are not just raw numbers but a number of some specified of units of measure. CSS has a variety of measurement units as outlined above. Each have benefits and appropriate usage.

See: https://www.w3.org/Style/Examples/007/units.en.html

Pixels are one of the most controllable units of measurement and have a fixed proportion on the screen as decided by the browser. It is not always a pixel on the screen but a length based measurement based upon the display. Consider a device such as the iPad3 retina displays with 326dpi resolution.

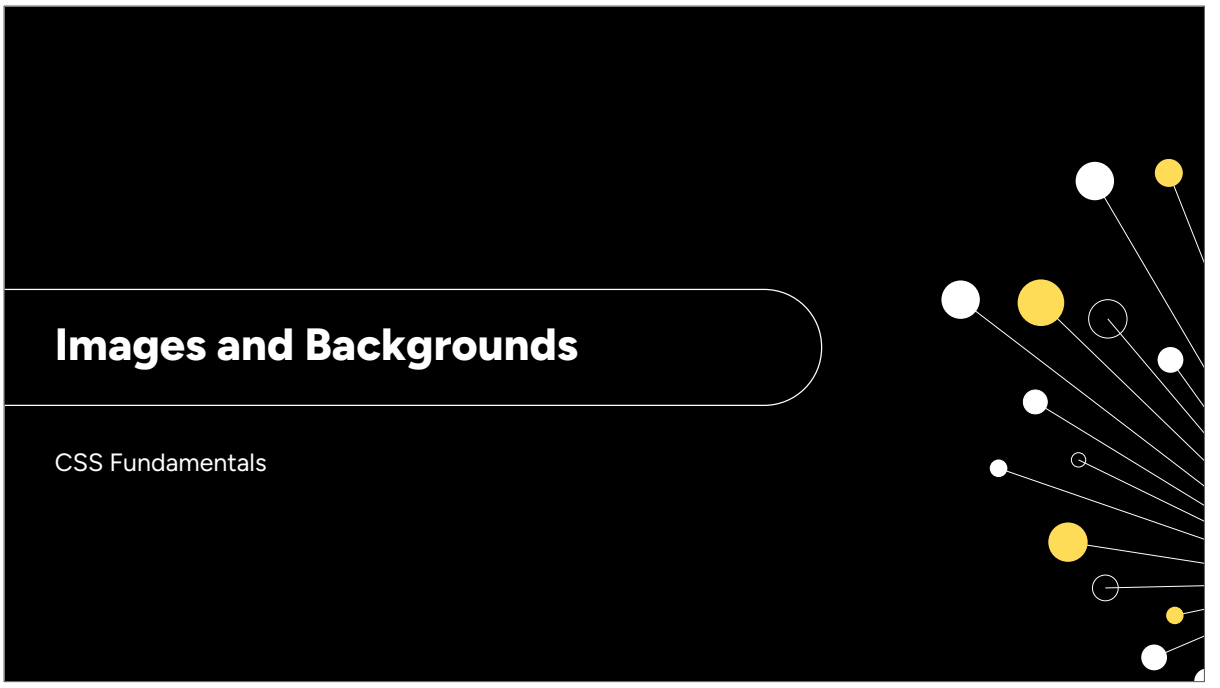
Ems are a relative unit. 1em is relative to the parent element's font size. It is very useful for keeping fonts in a proportion to each other in a display

Points are an absolute length-based unit equal to 1/72 of an inch. Points can be useful when setting types for print. They are rarely suitable for screen where points are approximated to the resolution of displays and system settings.

QA

Quick Lab 7 – Text, Colours and Sizing

Experiment with adding colours and sizing to text and elements


Images and Backgrounds
CSS Fundamentals

QA

Setting Image Properties

The markup for an image often contains the height and width

```
<img src="electric.JPG" width="450" height="372" />
```

Working towards multi device display we should avoid this

- Creates a hard coded appearance rule for a graphic

Attributes such as width and height can be set in the CSS

```
<img id="electric"
src="electric.JPG"/>
```

```
#electric {
            width: 450px;
            height: 372px;
}
```

Positioning, borders and spacing must be done with the box model

- Never use inline attributes

When creating images in HTML you may previously have set attributes define property such as a width or height into the HTML. This is not recommended a good practice web development as it constrains the image to a constant proportion. These properties can be overridden or updated using CSS and preferably we should place all appearance related behaviour within a suitable CSS selector. This becomes extremely important when we use media queries as we will be able to set attributes for different screen resolutions and the CSS will choose the correct settings for the correct screen proportions.

Another solution to this comes from using a srcset to allow the browser to choose which image to download.

QA

Page Background Colour and Images

The background can be a colour fill

- Use background-color CSS property of the <body> tag

```
<body style="background-color: #FFFFFF;">
<!-- White background -->
```

Alternatively, you can tile an image

- Use background-image CSS property of the <body> tag

```
<body style="background-image: url('paper.jpg');">
<!-- Background of paper texture -->
```

- Can also set whether to repeat and position

The **background-color** CSS property of the **<body>** tag sets the background colour of the page. The colour can be either a hexadecimal, red-green-blue colour value, a predefined colour name or a decimal rgb value.

The **background-image** CSS property of the **<body>** tag specifies a URL for a background picture. The picture can be tiled behind the text and graphics on the page. There are additional CSS settings that can specify whether or not to repeat the image and also set a specific position.

One very common technique is to define a very short, thin background GIF (typically 2000 pixels long and a few pixels high). When tiled across the screen, this will give the appearance of being tiled vertically but not horizontally.

QA

Element Background Images

The background of an element can be set using the url property

- The CSS requests an image asset using the url property

```
background-image: url(../img/thumb/mountain.jpg);
```

The following properties can also be set:

- **`repeat`**
  - Sets whether the image tiles appear only once
  - Or repeat only horizontally or vertically
- **`attachment`**
  - Sets whether the image scrolls with the rest of the page or stays in one place
- **`position`**
  - Moves the image to the left and down (positive values) or to the right and up (negative values)
  - Calculated from the top-left corner of the parent element.

Commonly used alternative for adding images to page is to set the background of a block element to use a graphic. There are pros and cons to this technique the major advantage is that the information for the graphic is embedded into the CSS this can reduce a number of header request required to process the image on the page. It also allows us to use advanced techniques such as spriting.

QA

Background Images

- **`size`**
  - Sets the width and height of the image within the element
    - As an absolute length or percentage
- **`clip`**
  - Sets if the background fits to the border or within the content area
- **`origin`**
  - Sets the position of the background relative to the border, padding, or content
- ***Multiple background images***
  - CSS3 allows you to layer multiple background images
    - Uses a comma-separated list

CSS3 has some extremely useful new properties for working with background images with the origin and size properties being especially useful. Allowing us to set a graphic that could scale automatically to fill the width of the containing element or a single larger graphic could be repositioned within the block saving on additional image requests from the server.

QA

So <Img> Or Background-image?

- Pros for **<img>**
  - Use **<img>** plus alt attribute if the image is part of the content
  - Use **<img>** when the image has an important semantic meaning, such as a warning icon.
    - This ensures that the meaning of the image can be communicated in all user-agents
    - Including screen readers.
  - Use **<img>** if you rely on browser scaling to render an image in proportion to text size.
  - Use **<img>** with a z-index in order to stretch a background image to fill its entire window.
  - Using **<img>** instead of background-image can dramatically improve performance of animations

The above checklist helps you determine the correct usage for an image or block background technique.

QA

So <Img> Or Background-image? - Pt2

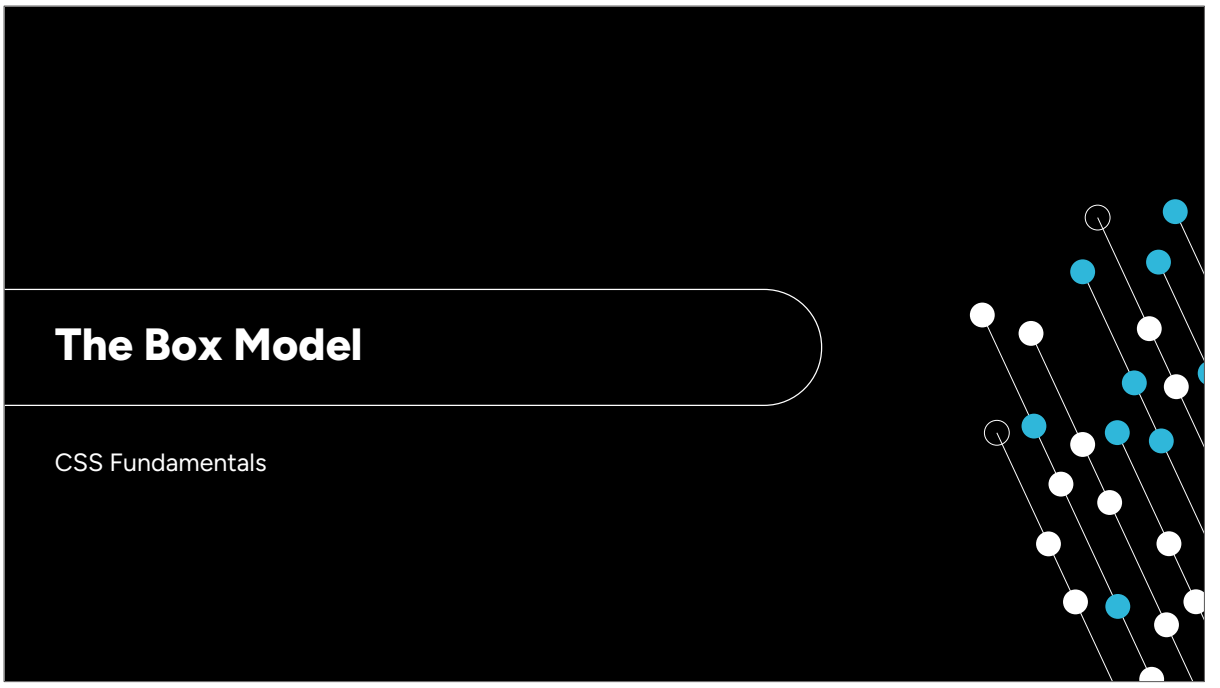
- Pros for CSS Background Images
  - Use CSS background images if the image is not part of the content.
  - Use CSS background images when doing image-replacement of text
  - Use background-image if you need to improve download times, as with CSS sprites.
  - Use background-image if you need for only a portion of the image to be visible
  - Use background-image if you need different images for different screen resolutions

The above checklist helps you determine the correct usage for an image or block background technique.

QA

Quick Lab 8 – Images and Backgrounds

- Experiment with the format of images on pages
- Add background images to elements and format how they are displayed


The Box Model
CSS Fundamentals

QA

The Box Model

All HTML elements can be considered as boxes

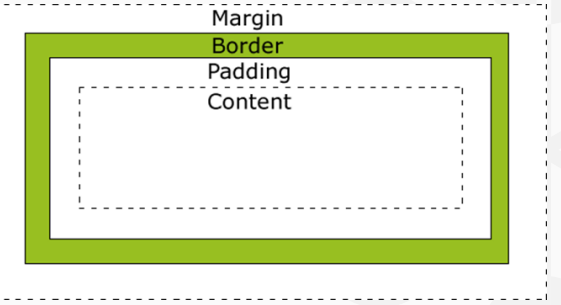
Box Model is used when talking about design and layout

Essentially a box that wraps around HTML elements

Consists of margins, borders, padding and the actual content

Box model allows:

- Placing a border around elements
- Space elements in relation to other elements



QA

The Box Model

Margin:

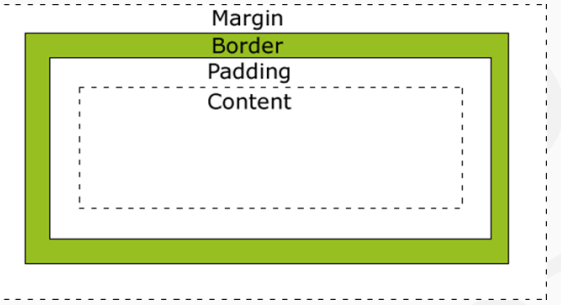
- Clears an area around the border
- Is transparent (no background colour)

Border:

- Goes around the padding and the content
- Affected by the background colour of the box

Padding:

- Clears an area around the content
- Affected by the background colour of the box



Content:

- The content of the box, where text and images appear

QA

The Box Model – Settings Properties

All HTML elements have four sides – top, bottom, left and right

- Properties can be set for each dimension or in a compound rule:

10px in all directions

```
margin: 10px;
margin: 10px 5px;
margin: 0px 5px 10px 15px;
margin-bottom: 2em;
```

10px top and bottom 5px left and right

Individual direction

top, right, bottom, left

Child elements typically have their own block properties

- Can be set independent of the parent
- The inner width of an element (content) available is a remainder of reserved space by parent elements
- Background colours and images can also be set

Elements have the following key properties:

- **Content** —At the centre of the box, this is the substance of the page. The content includes all text, lists, forms, and images
- **Child Elements** —Elements contained within the parent element that are set by their own HTML tags. Child elements typically have their own box that can be controlled independently of the parent.
- **Width and Height** —The dimensions of the box. If you leave width and height undefined, these dimensions are determined by the browser
- **Padding** —The space between the border and the content of the element. Background colours and images also fill this space. If left unset, the size of the padding is generally 0.
- **Background** —Space behind the content and padding of the element. This can be transparent, a solid colour, one or more background images, or a background gradient.
- **Border**—A rule (line) that surrounds the element and can be set separately on any of the sides. The border is invisible unless you set its colour, width, and style—solid, dotted, dashed, and so on. You can also set a border image. If left unset, the border size is generally 0.
- **Margin**—The space between the border of the element and other elements in the window. If left unset, the browser defines the margin.

QA

Element Width and Height

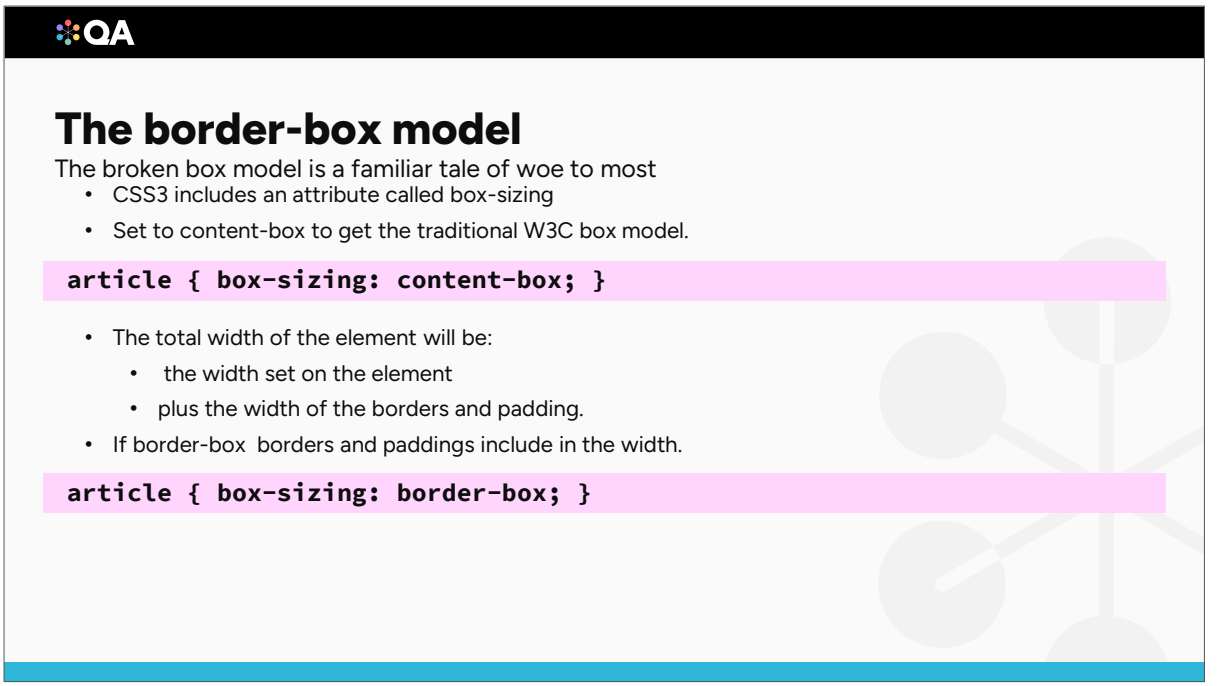
- When you set the width and height properties of an element with CSS you only set them for the **CONTENT** area
- To calculate the full size of an element you must add the padding, borders and margin to the width of the content
- ```
  width: 250px;
  padding: 10px;
  border: 5px solid gray
  margin: 10px
  ```

It is **300px**

Let's do the maths

+ 250px (content width)
+ 20px (left and right padding)
+ 10px (left and right border)
+ 20px (left and right margin)

300px
  ```



The age-old problem of having to use the conventional Level 2.1 box model in conjunction with padding and/or border values is solved using CSS3. Up until now, this problem was a major stumbling block for developers, particularly in the instance of specifying a border/padding value in relation to a fluid length element, but the new 'box-sizing' property answers this problem.

There are currently two values in the official spec- 'content-box' and 'border-box':-

'content-box' – "This is the behaviour of width and height as specified by CSS2.1. The specified width and height (and respective min/max properties) apply to the width and height respectively of the content box of the element. The padding and border of the element are laid out and drawn outside the specified width and height."

'border-box' – "The specified width and height (and respective min/max properties) on this element determine the border box of the element. That is, any padding or border specified on the element is laid out and drawn inside this specified width and height. The content width and height are calculated by subtracting the border and padding widths of the respective sides from the specified 'width' and 'height' properties."

There are browser specific hacks for this rule:

-moz-box-sizing: border-box;
-webkit-box-sizing: border-box;
box-sizing: border-box;

QA

Borders

Borders can have the following attributes set:

- `border-width: all [top, right, bottom left]`
- `border-style: all [top and bottom, left and right]`
- `border-color: top [left, bottom, right]`
  - Properties can be set individually for all by using shorthand **`border`** property

```
div { border: 2px dashed blue; }
```

Can specify **`border`** for each side by inserting **`top`**, **`left`**, **`bottom`** or **`right`** between **`border`** and *property to set* or use the shorthand:

```
div { border-top-style: double; }
div { border-left: 5px inset purple; }
```

QA

Rounded borders

- Pre-CSS3 had to be achieved through JavaScript or images:

```
border-radius: 30px
```

```
border-top-left-radius: 50px;
border-top-right-radius: 30px;
border-bottom-right-radius: 50px;
border-bottom-left-radius: 30px;
```

- Shorthand

```
border-radius: 50px 30px 50px 30px;
```

QA

Outline

Renders a uniform line for viewers attention

- Rendered on top of an elements rendering box
- Does not influence a box's position or size

```
outline: 3px dashed #3a5c7a;
```

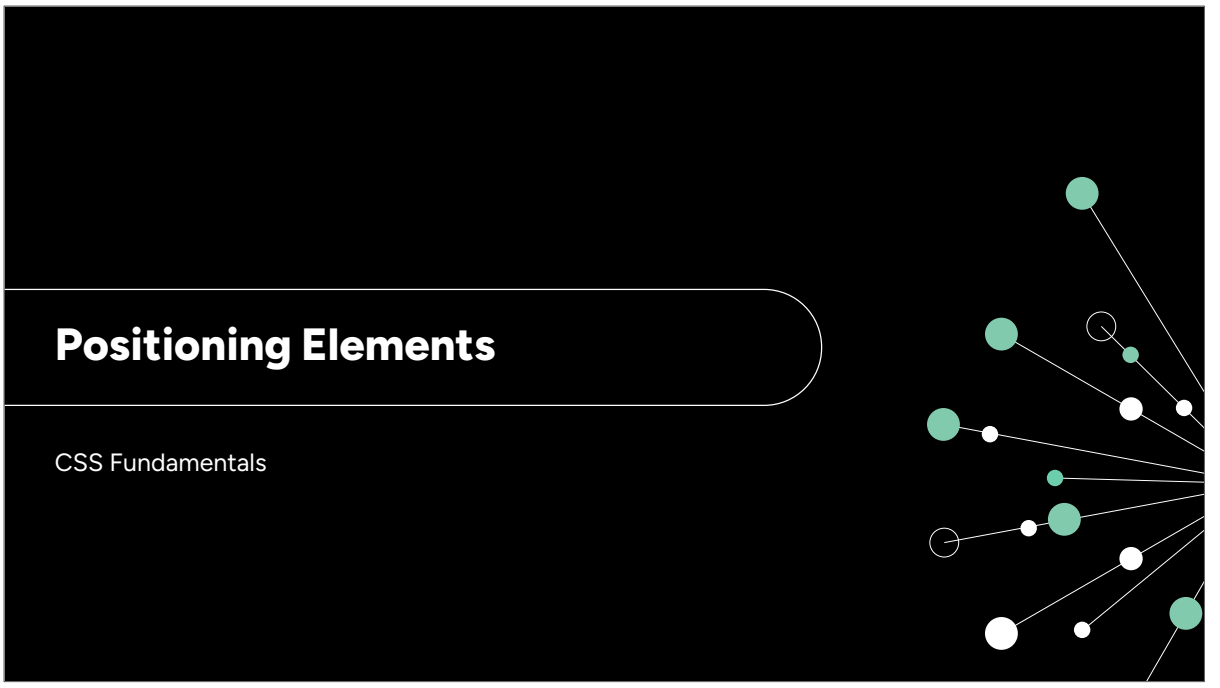
Optional outline-offset property

- Offsets an outline
- Then draws it beyond the border edge.

```
outline-offset: 10px
```

The 'outline-style' property specifies an outline line style for the current element.

The outline properties create a uniform line around an object in order to draw visual attention. An outline is slightly different than a border in several ways: An outline is drawn starting "just outside the border edge" and is allowed to be non-rectangular. Outlines are always rendered on TOP of an element's rendering box and do not influence the box's position or size calculation; the document does not need to be re-flowed when a border is rendered or hidden, but the outline may overlap other nearby elements


Positioning Elements
CSS Fundamentals

QA

Positioning Elements

- **Position: relative | static**
  - The content edge of the nearest block-level ancestor
- **Position: absolute**
  - The nearest positioned ancestor according to
    - The padding edge of the if the ancestor is block-level
    - The content edge of the first/last box if the ancestor is inline
- **Position: fixed**
  - The window / printed page

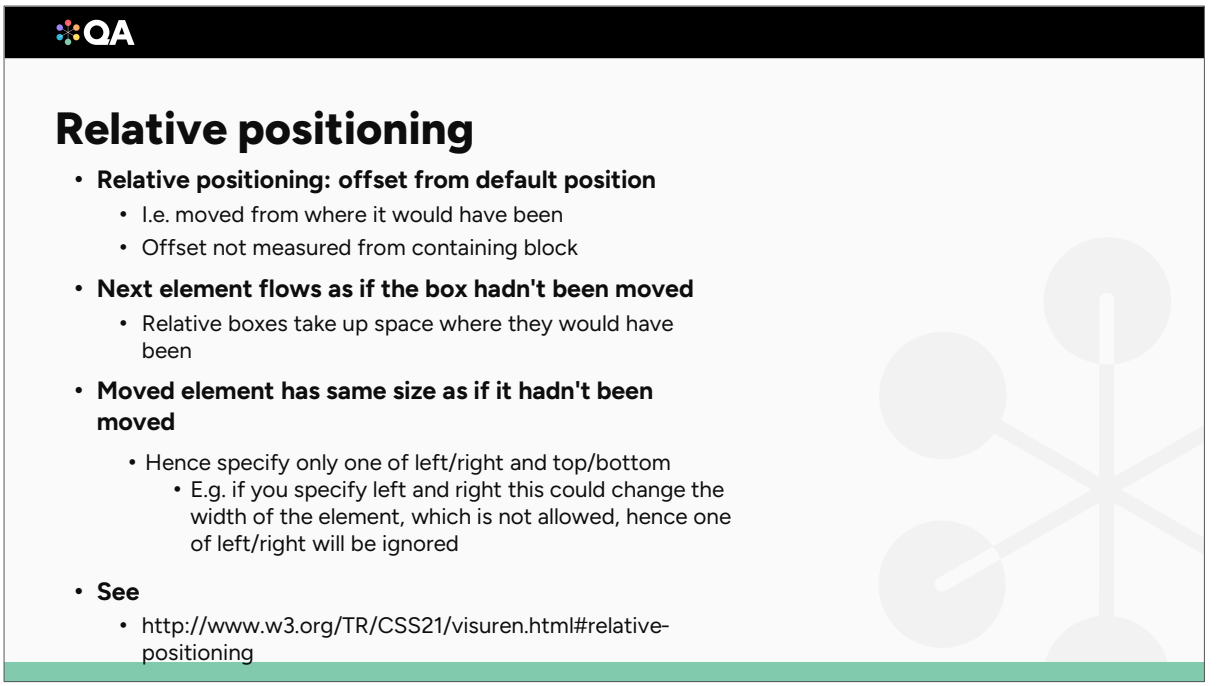
The values of the 'position' property have the following meanings:

static The box is a normal box, laid out according to the normal flow. The 'top', 'right', 'bottom', and 'left' properties do not apply.

relative The box's position is calculated according to the normal flow (this is called the position in normal flow). Then the box is offset relative to its normal position.

absolute The box's position (and possibly size) is specified with the 'top', 'right', 'bottom', and 'left' properties. These properties specify offsets with respect to the box's containing block. Absolutely positioned boxes are taken out of the normal flow. This means they have no impact on the layout of later siblings.

fixed The box's position is calculated according to the 'absolute' model, but in addition, the box is fixed with respect to some reference. As with the 'absolute' model, the box's margins do not collapse with any other margins. In the case of handheld, projection, screen, tty, and tv media types, the box is fixed with respect to the viewport and doesn't move when scrolled. In the case of the print media type, the box is rendered on every page, and is fixed with respect to the page box, even if the page is seen through a viewport (in the case of a print-preview, for example).



Relative positioning

Once a box has been laid out according to the normal flow or floated, it may be shifted relative to this position. This is called relative positioning. Offsetting a box (Box1) in this way has no effect on the box (Box2) that follows: Box2 is given a position as if Box1 were not offset and Box2 is not re-positioned after Box1's offset is applied. This implies that relative positioning may cause boxes to overlap. However, if relative positioning causes an 'overflow:auto' or 'overflow:scroll' box to have overflow, the browser must allow the user to access this content (at its offset position), which, through the creation of scrollbars, may affect layout.

A relatively positioned box keeps its normal flow size, including line breaks and the space originally reserved for it.

QA

Absolute Positioning

- **Absolute positioning: offset from container's position**
  - I.e. relative to container not page
- **Offset measured from**
  - Block level ancestor: the top, left of ancestor's padding box
    - I.e. outside of padding, inside of border
  - Inline ancestor: the top, left of the ancestor's content box
    - I.e. outside of content
- **See**
  - http://www.w3.org/TR/CSS21/visuren.html#position-props
  - http://www.w3.org/TR/CSS21/visuren.html#absolutely-positioned

Absolute positioning

In the absolute positioning model, a box is explicitly offset with respect to its containing block. It is removed from the normal flow entirely (it has no impact on later siblings). An absolutely positioned box establishes a new containing block for normal flow children and absolutely (but not fixed) positioned descendants. However, the contents of an absolutely positioned element do not flow around any other boxes. They may obscure the contents of another box (or be obscured themselves), depending on the stack levels of the overlapping boxes.

QA

Margin - Positive and Negative Values

Giving CSS positive values for padding or margin puts space between element and its reference

- Puts 20 pixels between the left margin of the element and its reference - effectively moves the element 20 pixels to the right

```
margin-left: 20px;
```

Giving CSS negative values for padding or margin moves the element towards its reference

- Effectively moves the element 20 pixels to the left

```
margin-left: -20px;
```

QA

Float and Clear

Float will move an element and flow text around it

- Treats the element as a block element and moves it left / right
- Rest of the page flows around the floated element
  - The available box is shrunk by the amount the floats take up

Clear will move an element to after the float

- Adds clearance to the top margin to move it clear of the float
  - Moves top border edge below the bottom outer edge of the float
  - Unless the cleared element is also a float (line up outer edges)

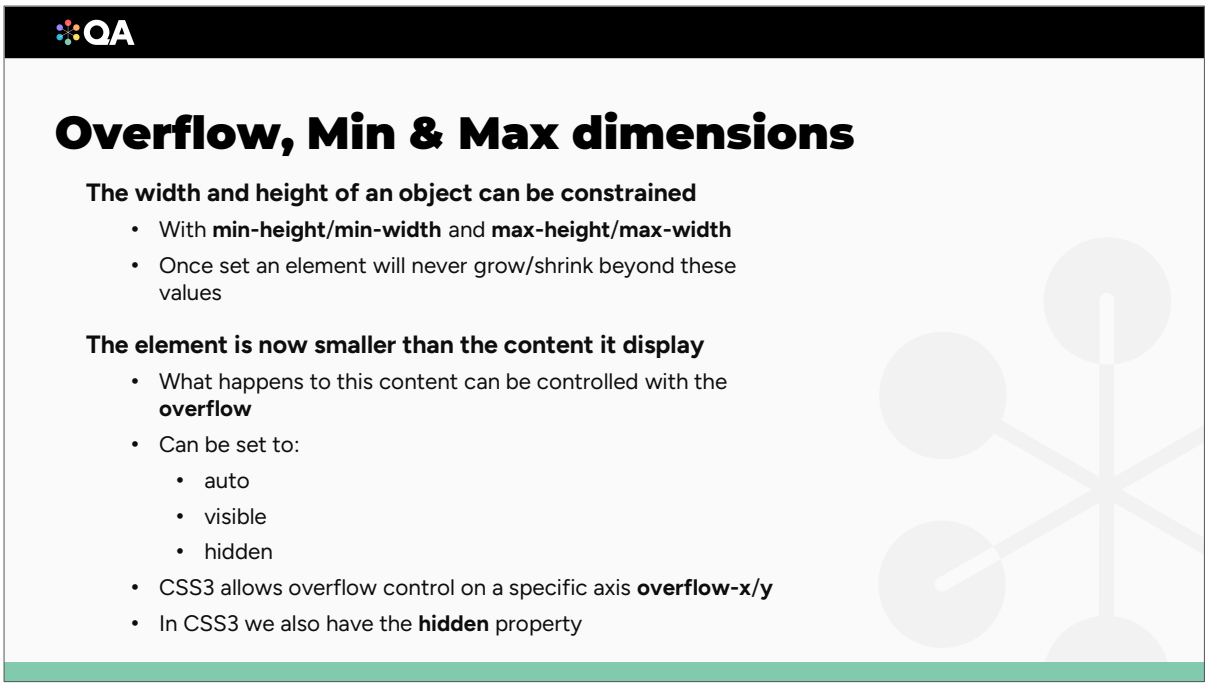
- **See**
  - http://www.w3.org/TR/CSS21/visuren.html#propdef-float
  - http://www.w3.org/TR/CSS21/visuren.html#propdef-clear

The 'float' property specifies whether a box should float to the left, right, or not at all. It may be set for any element, but only applies to elements that generate boxes that are not absolutely positioned. The values of this property have the following meanings:

left The element generates a block box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the 'clear' property).

right Similar to 'left', except the box is floated to the right, and content flows on the left side of the box, starting at the top.

none The box is not floated



An element can have a minimum and maximum width set as properties. Once set the element will never grow wider or narrower than those values.

Once you change the maximum width of an object content within the object may become clipped you can control what happens to this overflowing content with the overflow property. You can set the overflow property to one of the three following options:

Visible: visible forces the crop party element to be displayed instructor the browser to ignore any cropping set elsewhere in the CSS. This will rarely be applied directly in your CSS instead it is likely to be properly added or removed using JavaScript or a pseudo-hover selector.

Hidden: hidden will crop the element provide no way to access the rest of the element content. Only use hidden when you are absolutely sure of the content of the block.

Scroll: scroll set scrollbars around the visible area allows visitors to scroll to the elements content scroll will display a scroll bar on the overflow block whether it is needed or not.

Auto: auto allows a browser to the side where the scrollbars need to be displayed. Auto is generally the best option especially using relative units such as fonts as it allows scrollbars to be displayed as is needed.

QA

Controlling how an element displays

Elements are primarily set to be block or inline as their display type

- This behaviour can be changed in CSS
- By modifying the display attribute
- By setting an element property **`display:none`** it is hidden
  - The element is then removed from the flow
  - Can be accomplished with a `hidden` attribute in HTML5
  - Alternatively there is the `visible` property
    - Does not remove the element from the document flow

Elements can also be switched between inline and block display

- Useful for advanced layout

As we know elements are usually displayed as in-line or block. Allowing the browser to know how to layout defined positioning of every element. This display property can be overridden in CSS allowing us to treat non-block elements such as a button as a block or vice versa. A 16 with block-based positioning during this chapter block takes up the width of its containing element in the next block is positioned beneath.

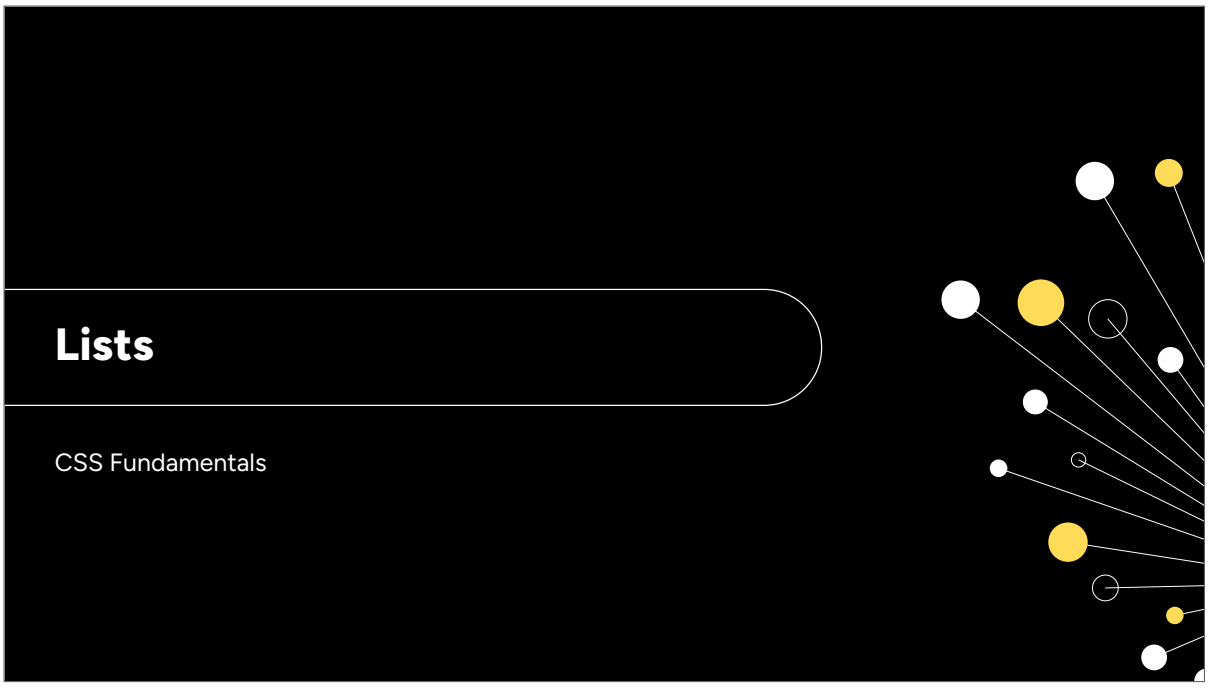
Traditionally the display property can also be set to none switching the display to none takes an element out of the flow of the document. There are additional display models provided by CSS which are less frequently used but important to understand.

As an additional piece of information HTML 5 provides a new global attribute called hidden. Hidden effectively has the same functionality as setting display: none. The hidden attribute is far more simple to set and remove using JavaScript or add is intended the declarative markup. For modern browsers this is the preferred approach that you would consider a fallback to display: none for legacy types

QA

Quick Lab 9 – Positioning Elements

Use positioning and styling techniques to layout a page to a given design


Lists
CSS Fundamentals

QA

List Styles

- Set on the enclosing list tag – either `<ul>` or `<ol>`
- Can be:

Property	Description	Examples of Possible Values
`list-style-image`	Sets an image as the list-item marker	`url("images/bullet.svg"), none`
`list-style-position`	Sets the position of the list-item markers	`inside, outside`
`list-style-type`	Sets the type of the list-item marker	`disc, circle, square, decimal, georgian, none, inherit, initial`
`list-style`	Shorthand that sets all properties in one declaration	`lower-roman url("images/bullet.svg") outside`

For a full list of values:

list-style-image: https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-image#Syntax

list-style-position: https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-position#Syntax

list-style-type: https://developer.mozilla.org/en-US/docs/Web/CSS/list-style-type#Syntax

list-style: https://developer.mozilla.org/en-US/docs/Web/CSS/list-style#Syntax

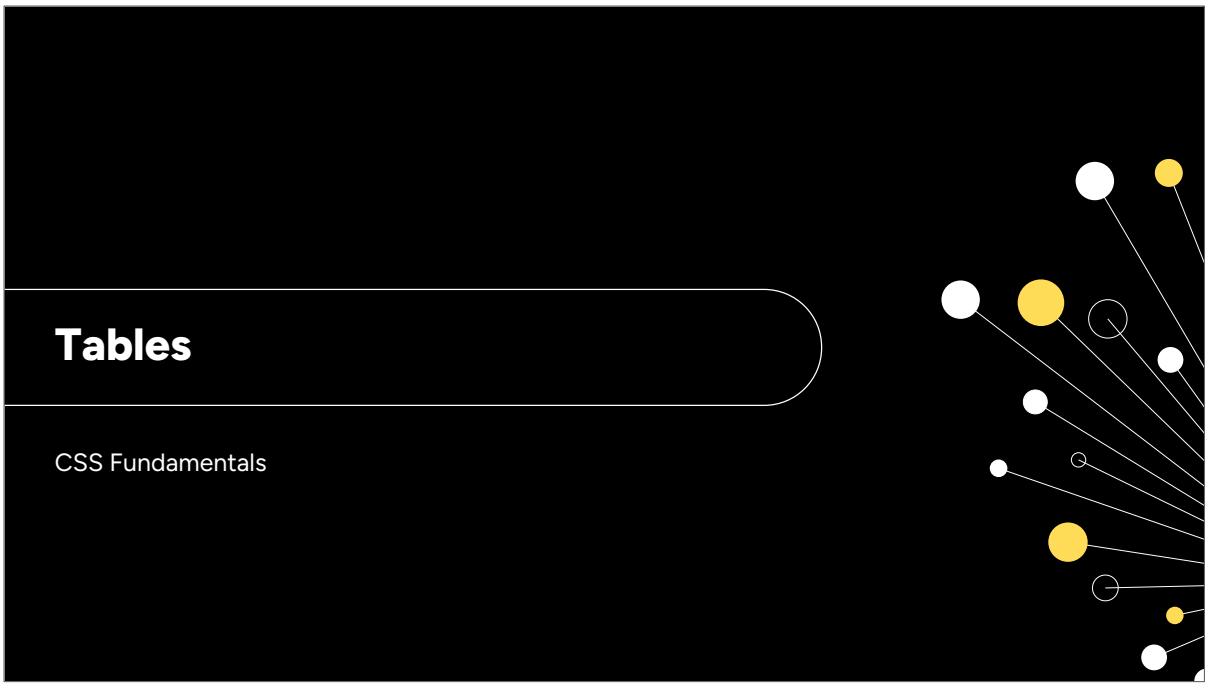
QA

Lists with Custom Counters

Useful for making outline lists

- New instance of counter automatically created in child elements
- Uses CSS function **`counters()`** – can insert separating text in between different levels

```
ol {
  counter-rest: section;            /* Creates new instance of section
                                       counter for each new ol element */
  list-style-type: none;
}
li::before {
  counter-increment: section;       /* Increments only this instance */
  content: counters(section, ". ") " ";   /* Combines values of all
                                             instances of section
                                             counter, separated by a . */
```


Tables
CSS Fundamentals

QA

Tables

Tables can be controlled with CSS with a series of properties

- The first is the `table-layout` which has two options that describe how to precisely divide up column widths
  - `auto`
  - `fixed`

Inter-cell padding is set with the `border-spacing` attribute - Equal in all directions

Every table cell defined by a `<td>` or `<th>` tag has four borders

- These butt up against each other so setting a `1px` border with no `border-spacing` the gap is doubled
- This can be controlled with the `border-collapse` property
  - `separate` – default borderers butt
  - `collapse` – borders overlap

Different browsers use different methods to calculate how a particular table should be displayed. There are two table-layout methods

- Fixed method bases its layout on the width of the table and the width of columns in the first row. This method is generally faster than automatic.
- Automatic uses the table column width along with the amount of content in the table data cell to calculate the table data cell width. This will generally render more slowly than the fixed method, but it also produces more accurate results for widths throughout the table.

The border-collapse property allows you to set a table so that each table data cell will share its borders with an adjacent table data cell rather than creating a separate border for each.

QA

Table Properties

- Set on the enclosing `<table>` tag
- Can be:

Property	Description	Examples of Possible Values
`caption-side`	Puts content of table's <caption> on specified side	`top, bottom`
`empty-cells`	Sets how browser should render borders and backgrounds around table cells that have no visible content	`show, hide`
`vertical-align`	Sets vertical alignment of an inline or table-cell box	`baseline, sub, super, text-top, text-bottom, middle, top, bottom`

For a full list of table properties and values see:

https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Table

QA

Table Formatting and Interactivity

The pseudo-class :hover can be applied to **<tr>**

- Will change the style of the row dependent on the format set

```
tr:hover { background-color: hotpink; }
```

Striped tables can be created by using the **nth-child** pseudo-selector and **odd** or **even**

```
tr:nth-child(odd) { background-color: palevioletred; }
```

Responsive tables can be created to display a horizontal scroll bar if the screen size is too small to display the whole content of the table

- Add a container around the table and use **overflow-x: auto**

```
<div style=overflow-x : auto>
  <table>…table content</table>
</div>
```

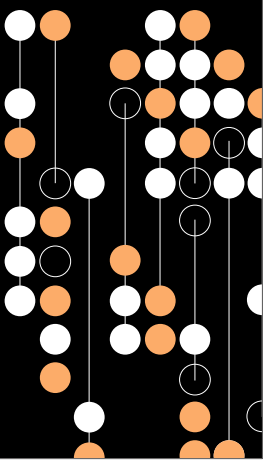
QA

Quick Lab 10 – Tables with CSS

Add styling to a table to make it more readable and interactive with hoverable rows

Animations, Transitions and Transformations

CSS Fundamentals



QA

@keyframe at-rule

Defines and controls the immediate steps in a CSS animation sequence

- Defines styles for keyframes along animation sequence – name used in **`animation-name`**
- Gives more control over immediate steps than transitions

```
@keyframes slidein {
  from {
    margin-left: 100%;
    width: 300%
  }
  to {
    margin-left: 0%;
    width: 100%
  }
}
```

QA

Animation Properties (1)

Allows animation of CSS properties over time using keyframes and properties below:

Property	Description	Examples of Possible Values
`animation-name`	Specifies one or more animations that should be applied to an element (defined by `@keyframes`)	`none, slide, bounce`
`animation-duration`	Sets length of time that animation takes to complete one cycle	`0s, 750ms`
`animation-timing-function`	Sets how animation should progress over duration of cycle	`Linear, ease-in-out, steps(5, end)`
`animation-delay`	Sets when animation should start – immediately, in the future or partway through the animation cycle	`250ms, -2s`

animation-name: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-name#Syntax

animation-durations: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-duration#Syntax

animation-timing-function: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-timing-function#Syntax

animation-delay: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-delay#Syntax

QA

Animation Properties (2)

Allows animation of CSS properties over time using keyframes and properties below:

Property	Description	Examples of Possible Values
`animation-iteration-count`	Specifies number of times animation should play before stopping	`0, 2, 3.2`
`animation-direction`	States whether animation should play forwards, backwards or alternate	`normal, reverse, alternate, alternate-reverse`
`animation-fill-mode`	Sets how animation should apply styles to target before and after	`none, forwards, backwards, both`
`animation-play-state`	Sets whether animation is playing or paused	`paused, running`

animation-iteration-count: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-iteration-count#Syntax

animation-direction: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-direction#Syntax

animation-fill-mode: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-fill-mode#Syntax

animation-play-state: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-play-state#Syntax

QA

Animation Properties (3)

Shorthand **`animation`** can be used to specify all properties:

```
animation: 3s ease-in 1s 2 reverse both paused slidein
```

Order:

- **`duration | timing-function | delay | iteration-count | direction | fill-mode | play-state | name`**

- Would run an animation that lasted for 3 seconds after a delay of 1 second, easing in, running twice in reverse starting paused and using the **`slidein`** definitio

PLAY

QA

Overview of Transitions

CSS3 allows you to define transitions for property changes

- E.g. when a user hovers over an element, change its size to XXX over a period of YYY
- The transition kicks in automatically on the property value changes

To define a simple transition in a CSS rule:

- Set the **transition** property
- Specify the property to vary and the duration of the transition

```
someCssRule {
  …
  transition: aProperty duration;
}
```

Note:

- You must use vendor-specific extensions for some browser versions

The simplest way to define a transition for a property is to set the **transition** CSS property in a CSS rule. Specify the following information:

- The target property that you want to define a transition for, e.g. **width**.
- The duration over which you want the transition to play out, e.g. **0.5s**.

The transition kicks in automatically whenever the target property is changed. You do not need to write any procedural code to trigger the transition! For example, the following circumstances would trigger a transition automatically:

- You have defined a CSS rule based on a pseudo-rule such as hovering over an element, and the user hovers over the element.
- You have written some JavaScript code somewhere that programmatically sets the target property to a new value.

Note: You must use vendor-specific extensions for some browser versions, as listed below. See http://caniuse.com/#search=CSS3%20transitions for details.

- -ms-transition: Microsoft Internet Explorer
- -webkit-transition: Google Chrome and Apple Safari
- -moz-transition: Mozilla Firefox
- -o-transition: Opera

QA

Transition Properties (1)

Enables definition of transition between 2 states of an element

- States may be defined using pseudo-classes or dynamically set using JavaScript

Property	Description	Examples of Possible Values
`transition-property`	Defines which CSS property (or properties) for transition	`margin-right, width, height`
`transition-duration`	Defines number of seconds or milliseconds a transition should take	`500ms, 2s`
`transition-timing-function`	Sets timing function to set intermediate values during transition	`Linear, ease-in, steps(6, end), cubic-Bezier(1, 1, 1, 1)`
`transition-delay`	Sets amount of time to wait before starting the transition	`250ms, 1s`

animation-name: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-name#Syntax

animation-durations: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-duration#Syntax

animation-timing-function: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-timing-function#Syntax

animation-delay: https://developer.mozilla.org/en-US/docs/Web/CSS/animation-delay#Syntax

QA

Transition Properties (2)

Shorthand **`transition`** can be used to specify all properties:

```
transition: margin-right 2s ease-in-out .5s
```

Order:

- **`property | duration | timing-function | delay`**

- Would run an transition that lasted 2 seconds after a delay of 0.5 seconds, easing in then out on the margin-right property of the element it has been applied to

QA

2D Transformations

CSS3 supports 2D and 3D transforms

- Enables elements rendered by CSS to be transformed in space

To define a transformation in a CSS rule:

- Set the **transform** property
- Optionally set the **transform-origin** property

```
someCssRule {
  …
  transform: transformation-function(s);
  transform-origin: horizPosition vertPosition;

}
```

To define a transformation in a CSS rule, set the **transform** property to a transformation function. The following 2D transformation functions are available:

- Translation: **translate()**, **translateX()**, **translateY()**
- Scaling: **scale()**, **scaleX()**, and **scaleY()**
- Rotation: **rotate()**, **rotateX()**, **rotateY()**
- Skew: **skew()**, **skewX()**, **skewY()**
- General transformation: **matrix()**
- No transformation: **none()**

By default, transformations are applied about the centre-point of the target element. If you want to change the transformation origin, you can set the **transform-origin** property and specify the horizontal position and vertical position for the transformation origin as follows:

- The horizontal position can be a length, percentage, or **left** / **center** / **right**.
- The vertical position can be a length, percentage, or **top** / **center** / **bottom**.

Note: You must use vendor-specific extensions for some browser versions for the **transform** and **transform-origin** properties, as follows:

- **-ms-transform** and **-ms-tranform-origin**: < Internet Explorer 10
- **-webkit-transform** and **-webkit-transform-origin**: Edge, Chrome,Safari, Firefox (see note for <44), Opera (15+)

QA

Transform Functions (1)

Different Transform functions – for Rotation:

Function	Description
`rotate()`	Rotates element around fixed point on 2D plane
`rotate3d()`	Rotates element around fixed axis in 3D space
`rotateX()`	Rotates element around horizontal axis
`rotateY()`	Rotates element around vertical axis
`rotateZ()`	Rotates element around z-axis

Different Transform functions – for Skewing

Function	Description
`skew()`	Skews element on 2D plane
`skewX()`	Skews element in horizontal direction
`skewY()`	Skews element in vertical direction

QA

Transform Functions (2)

Different Transform functions – for Scaling:

Function	Description
`scale()`	Scales element up or down 2D plane
`scale3d()`	Scales element up or down in 3D space
`scaleX()`	Scales element up or down horizontally
`scaleY()`	Scales element up or down vertically
`scaleZ()`	Scales element up or down along z-axis

Different Transform functions – for Matrix Transformations

Function	Description
`matrix()`	Describes a homogeneous 2D transformation matrix
`matrix3d()`	Describes a 3D transformation as a 4x4 homogeneous matrix

QA

Transform Functions (2)

Different Transform functions – for Translation:

Function	Description
`translate()`	Translates element on 2D plane
`translate3d()`	Translates element in 3D space
`translateX()`	Translates element horizontally
`translateY()`	Translates element vertically
`translateZ()`	Translates element along z-axis

Different Transform functions – for Perspective

Function	Description
`perspective()`	Sets distance between the user and the z=0 plane

QA

Translations

To translate an element, use one of these CSS functions:

```
translate(tx, [ty])
```

```
translateX(tx)
```

```
translateY(ty)
```

Example:

```
someCssRule {
  transform: translate(400px, 20px);
}
```

To define a translation transformation, set the **transform** property to one of the following functions:

- **translate(tx, [ty])**
- **translateX(tx)**
- **translateY(ty)**

tx specifies a horizontal translation. You can specify an absolute value or a percentage of the width of the target element. Note the following points:

- If **tx** is negative, it represents a translation to the left.
- If **tx** is positive, it represents a translation to the right.

ty specifies a vertical translation. You can specify an absolute value or a percentage of the height of the target element. Note the following points:

- If **ty** is negative, it represents a translation upwards.
- If **ty** is positive, it represents a translation downwards.
- If you call **translate()** and omit **ty**, the default value for **ty** is **0**.

QA

Learning Objectives

- Understand what CSS is
- Understand how CSS can be applied to web pages
- Understand the syntax of writing CSS rules
- Be able to select elements to apply CSS to
- Be able to work with Text, Colours and Images
- Be able to work with the Box Model and position elements
- Be able to style lists and tables
- Be able to add CSS animations, transforms and transitions to elements