Web Fundamentals

CSS

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

CASCADING STYLE SHEETS

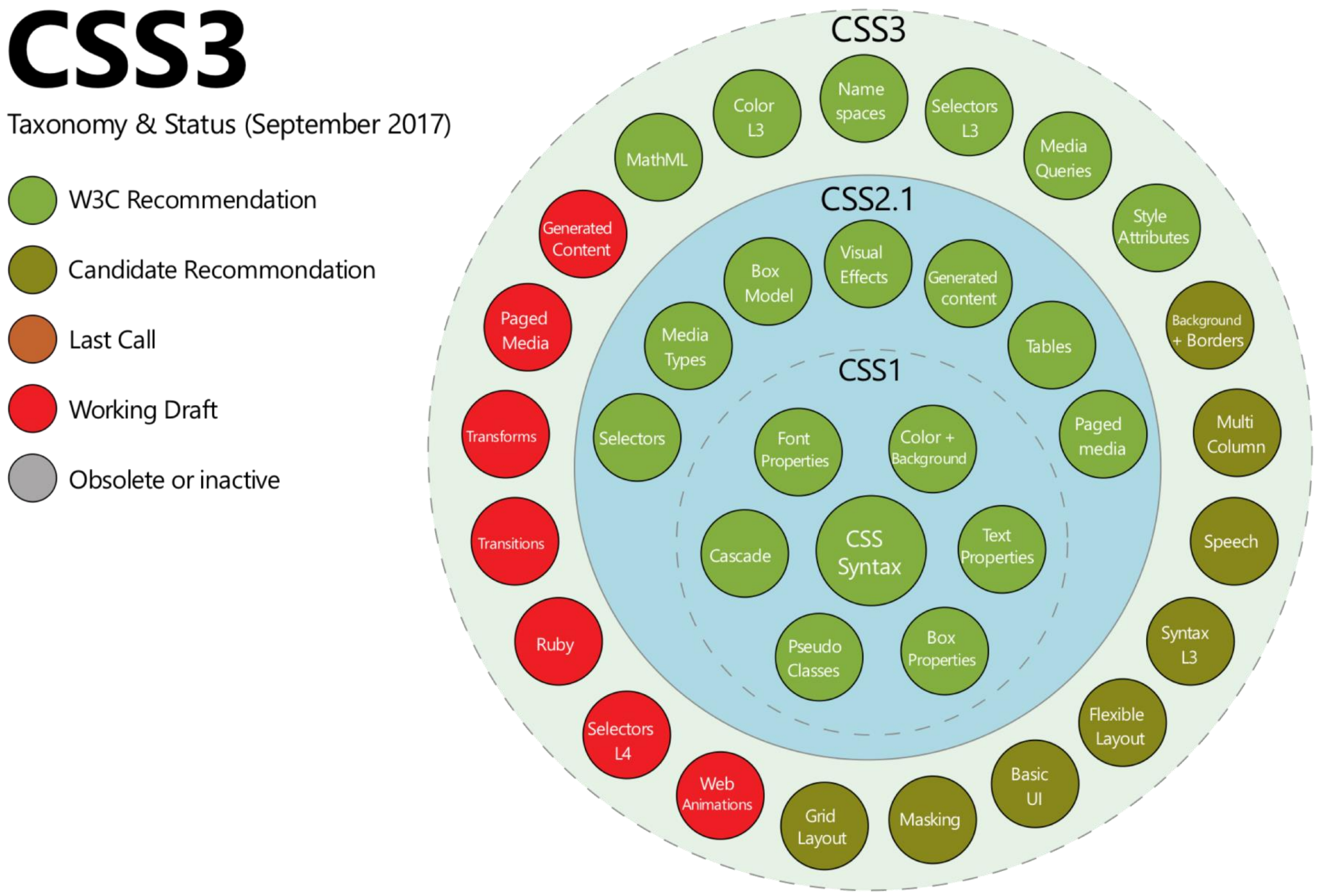
- CSS application and syntax
- CSS and the DOM
- Inheritance and Selecting Elements
- Text and Colours
- Measurement Units
- Images and Backgrounds
- The Box Model
- Lists and Tables
- Animations, Transitions and Transformations

CSS Application and Syntax

CSS FUNDAMENTALS

Cascading Style Sheets

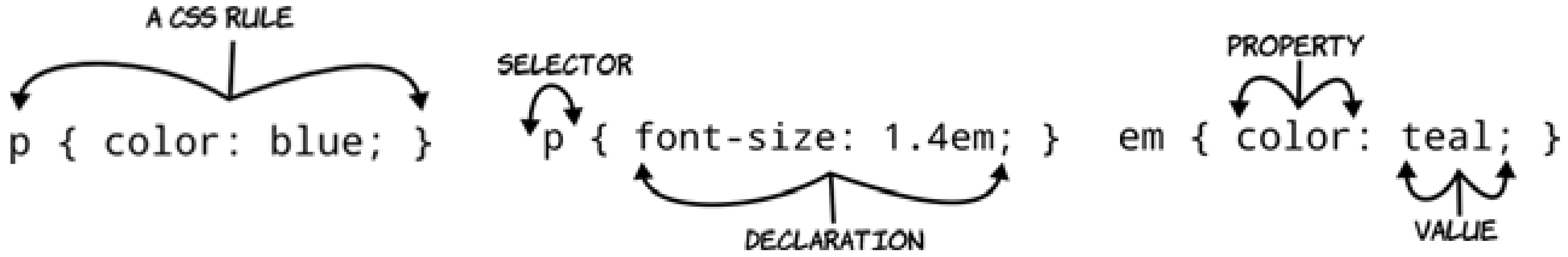
- CSS stands for Cascading Style Sheets
- It describes how HTML elements are to be displayed
  - This could be on the screen, on other media or even how it should be printed on paper
- Can control the layout of multiple web pages from a set of rules
- Styling can be applied in one of four ways:
  - Inline – 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)



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

CSS Basic Syntax

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



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>
```

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>
.. .. ..
```

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

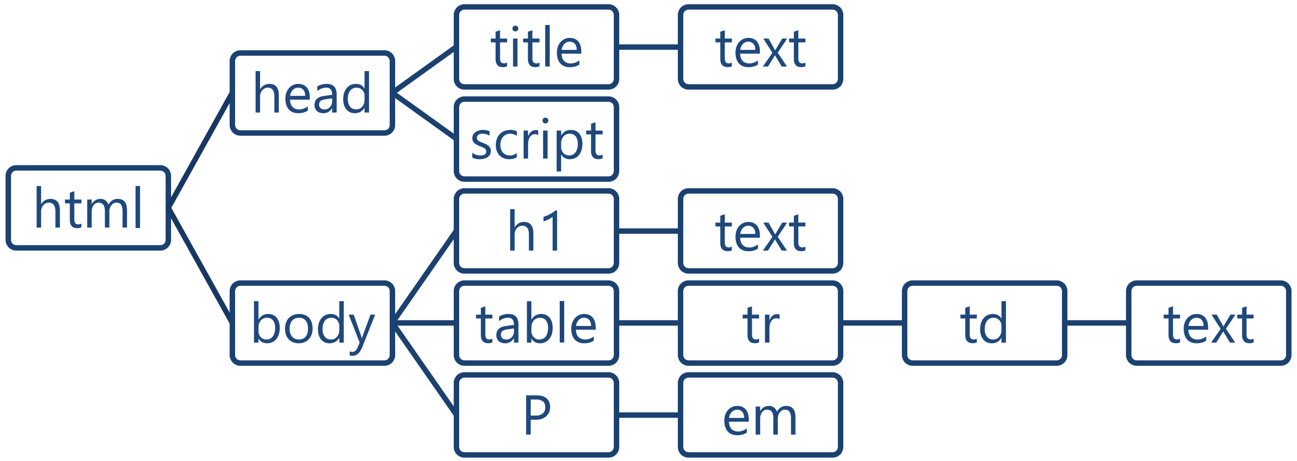
```
<link href="styles.css" rel="stylesheet">
```

CSS and the DOM

CSS FUNDAMENTALS

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.

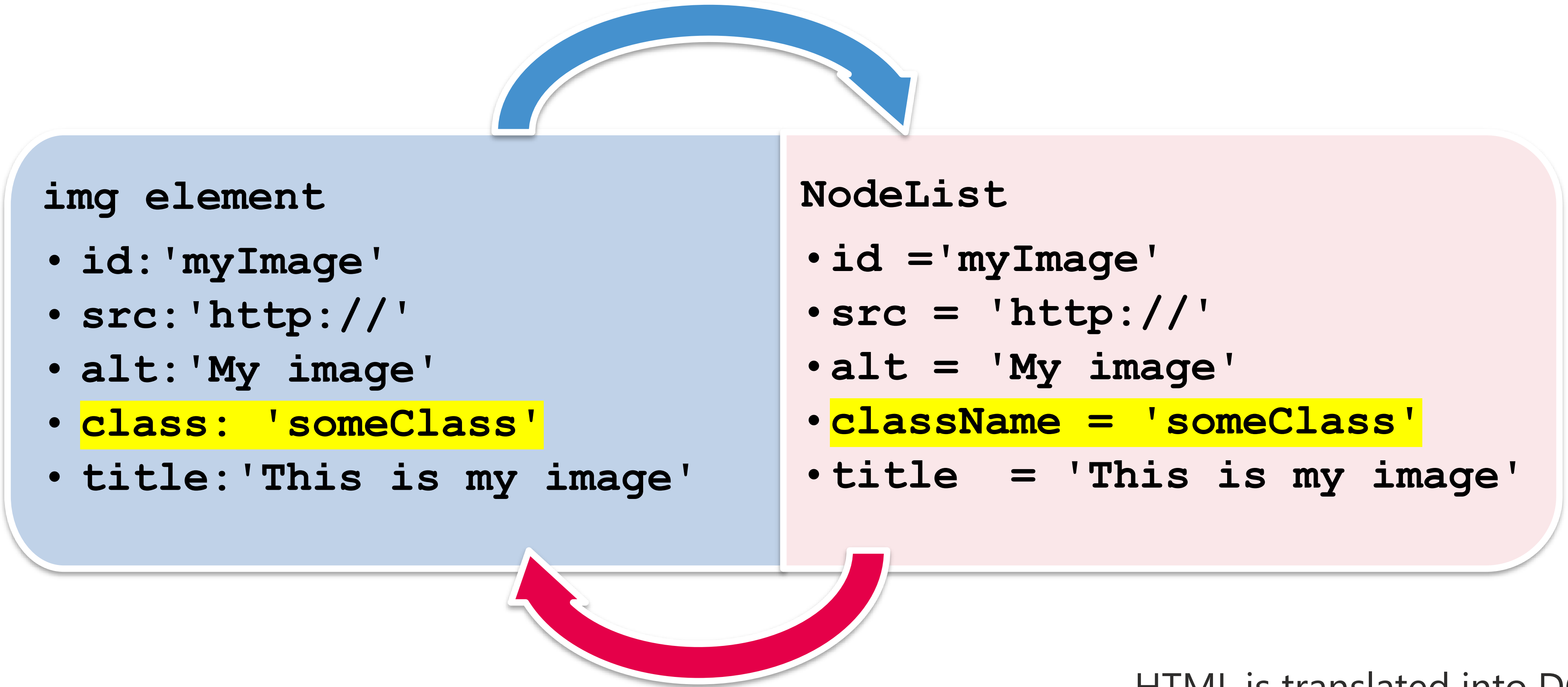
HTML markup to DOM object (1)

- 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 MARKUP TO DOM OBJECT (2)



img element

- **id:'myImage'**
- **src:'http://'**
- **alt:'My image'**
- **class: 'someClass'**
- **title:'This is my image'**

NodeList

- **id ='myImage'**
- **src = 'http://'**
- **alt = 'My image'**
- **className = 'someClass'**
- **title = 'This is my image'**

HTML is translated into DOM elements, including the attributes of the tag and the properties created from them.

These can be used to select and style elements

Inheritance and Selecting Elements

CSS FUNDAMENTALS

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

Selecting all tags of a particular type

- This is as simple as creating a rule for the tag name and nothing else
- All elements with this tag will be affected, regardless of where they are in the DOM
    - ***Assuming that this is the only CSS applied to the page (more on that later…)***

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

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

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

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

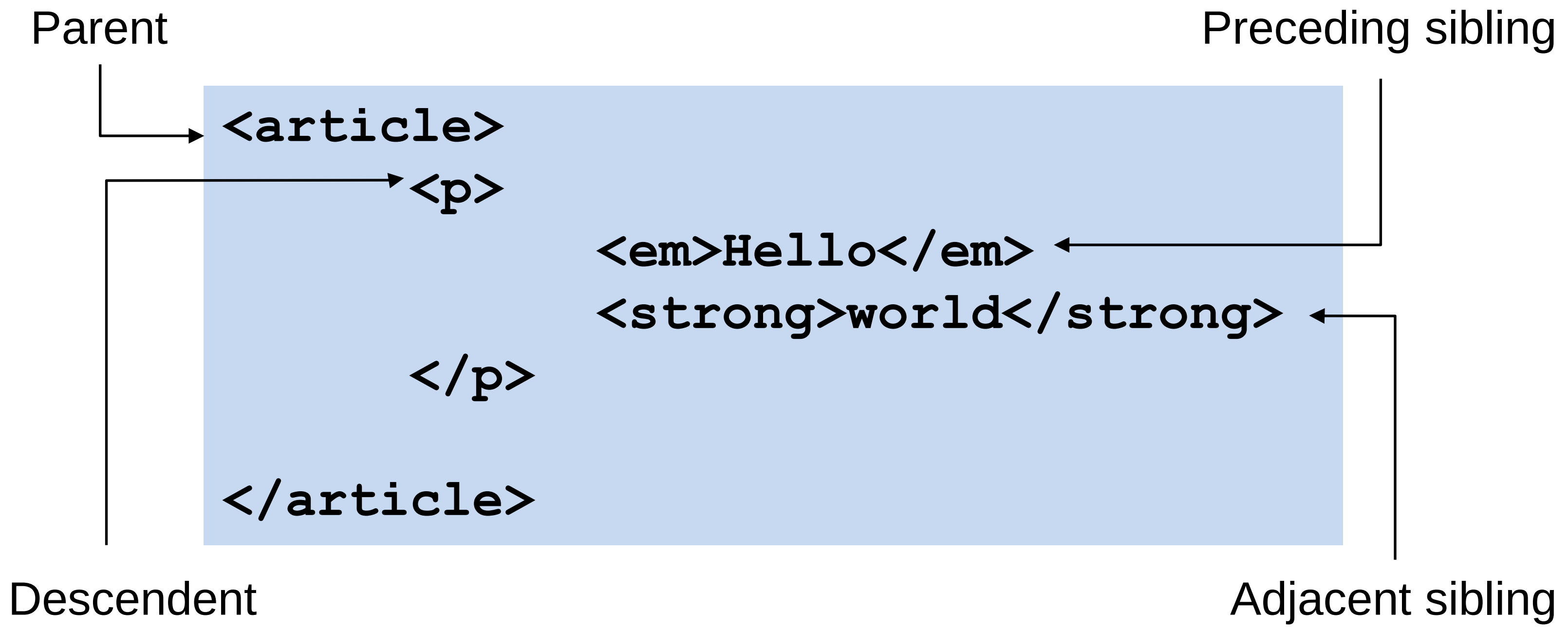
- Multiple elements can be selected by putting a comma between them

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

- Would make the text of *any* `h1` element and *any* `a` element **pink**

Understanding Inheritance

- 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.

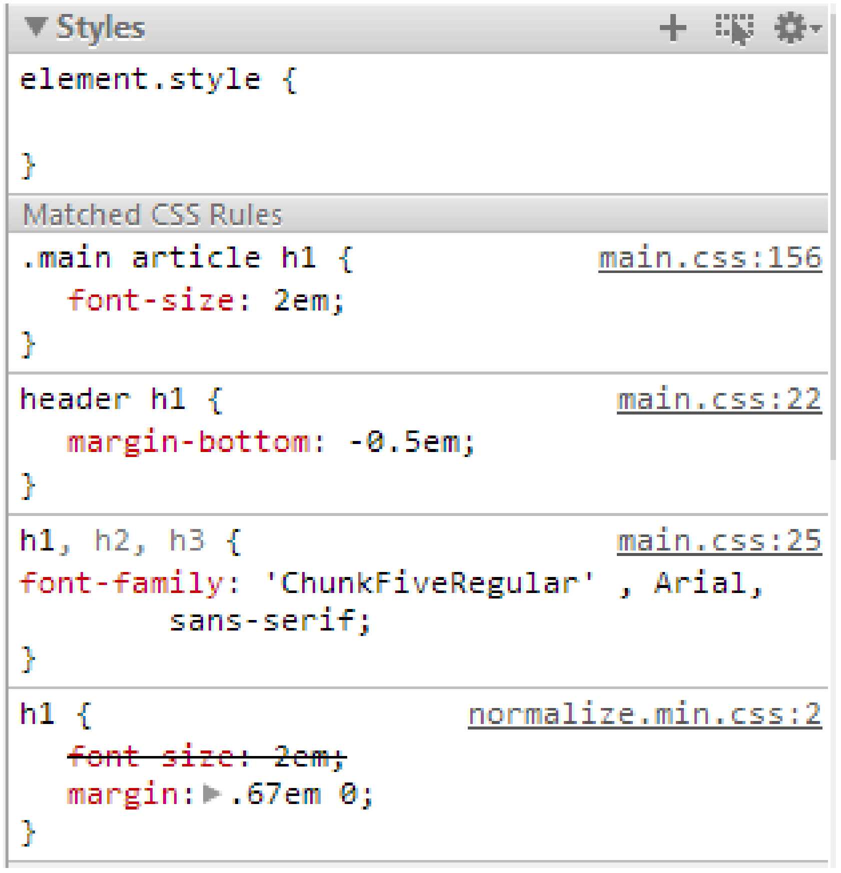


```
<article>
      <p>
            <em>Hello</em>
            <strong>world</strong>
      </p>

</article>
```

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



```
element.style {

}
.main article h1 {                main.css:156
   font-size: 2em;
}
header h1 {                        main.css:22
   margin-bottom: -0.5em;
}
h1, h2, h3 {                       main.css:25
font-family: 'ChunkFiveRegular' , Arial,
        sans-serif;
}
h1 {                      normalize.min.css:2
   ~~font-size: 2em;~~
   margin:▶ .67em 0;
}
```

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*

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>
```

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; }
```

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;
}
```

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; }
```

Pseudo classes specificity

- Recall: in the cascade styles are sorted by specificity
  - Latter rules are more specific that earlier rules

- Hence for link pseudo classes to work use this order

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

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;
}
```

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; }
```

Quick Lab 6 – CSS Selectors

- Apply selectors to style rules to apply styling to particular elements

Text and Colours

CSS FUNDAMENTALS

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 attacks

�

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

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.

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;
```

- With the exception of a sneakily inserted line-height
    - Note the lack of measurement unit
    - You can add them but it can cause issues

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

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`

Adding a drop shadow

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

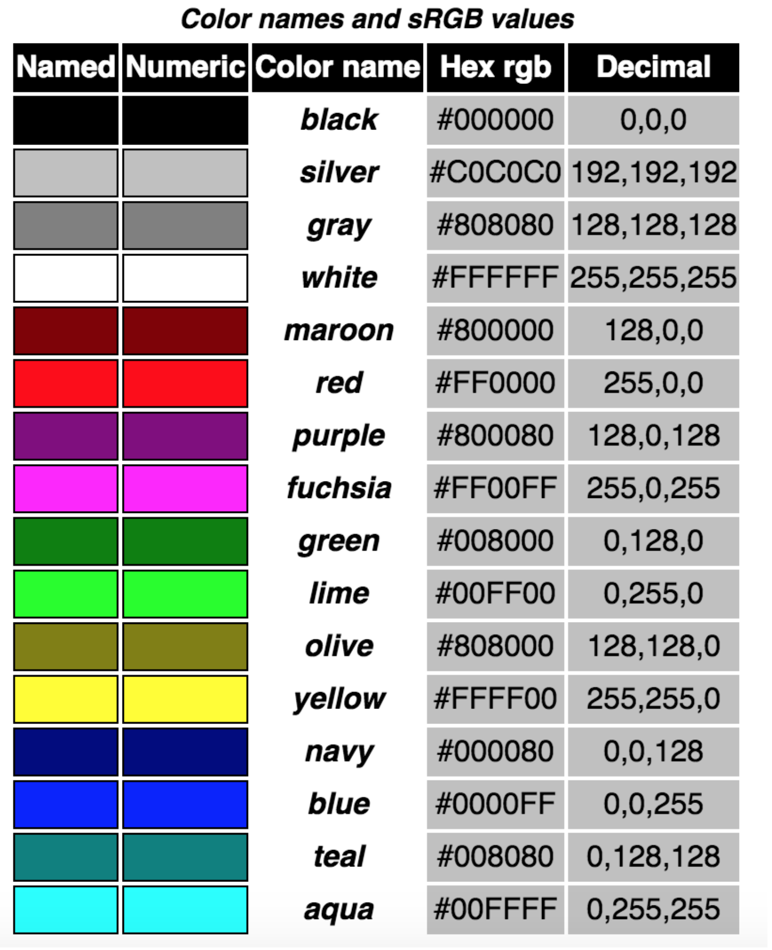
```
.shadow {
      text-shadow: 10px 8px 20px rgb(56, 52, 153);
}
```

- 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);
}
```

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



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

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 */
```

color values and format – HSL/HSLA

- RGB is hardware oriented and harps back to the days when CRT were used in monitors
- HSL are encoded as Hue, Saturation and Lightness
    - Hue is represented as an angle of the colour circle – measured in degrees and value is used in CSS
    - 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

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%; }
```

Quick Lab 7 – Text, Colours and Sizing

- Experiment with adding colours and sizing to text and elements

Images and Backgrounds

CSS FUNDAMENTALS

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

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

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.

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

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

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

Quick Lab 8 – Images and Backgrounds

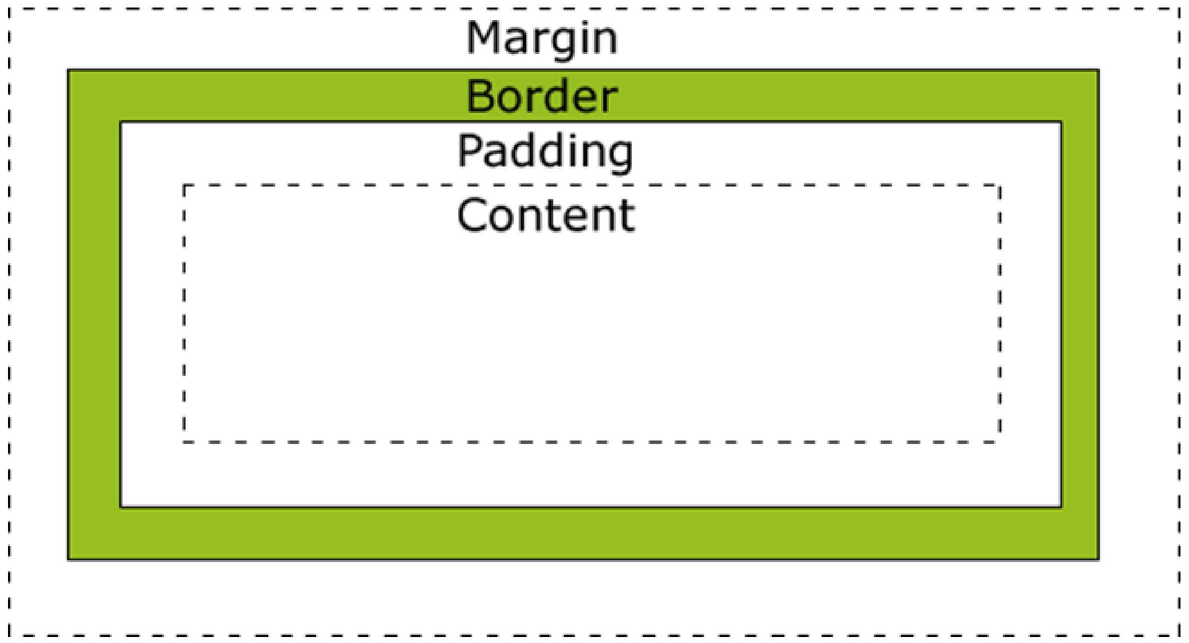
- Experiment with the format of images on pages
- Add background images to elements and format how they are displayed

The Box Model

CSS FUNDAMENTALS

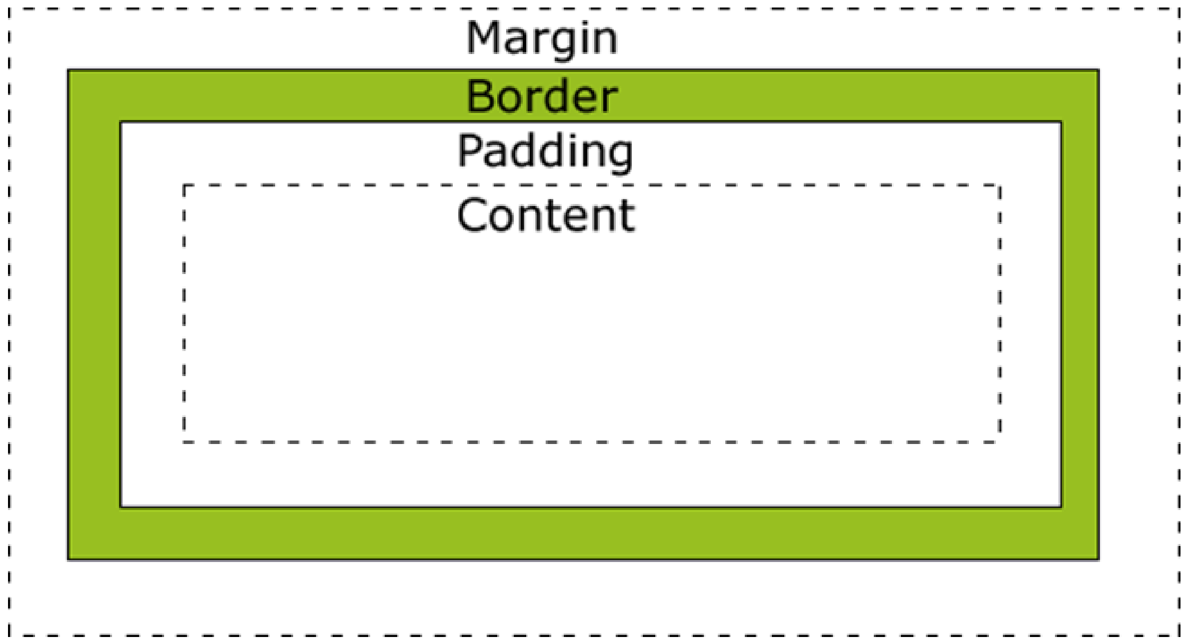
The Box Model

- 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



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



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

`margin: 10px 5px;` ← 10px top and bottom 5px left and right

top, left, bottom right → `margin: 0px 5px 10px 15px;`

`margin-bottom: 2em;` ← Individual direction

- 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

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
- What is the **TOTAL** width of the space the element takes here?

```
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 border-box model

- The broken box model is a familiar tale of woe to most
    - CSS3 includes an attribute called box-sizing
    - Set to content-box to get the traditional W3C box model.

```
article { box-sizing: content-box; }
```

- The total width of the element will be:
    - the width set on the element
    - plus the width of the borders and padding.
- If border-box borders and paddings include in the width.

```
article { box-sizing: border-box; }
```

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; }
```

Rounded borders

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

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

- Different radius can be added to different corners

```
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;
```

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
```

Positioning Elements

CSS FUNDAMENTALS

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

Relative positioning

- Relative positioning: offset from default position
  - I.e. moved from where it would have been
  - Offset not measured from containing block
- Next element flows as if the box hadn't been moved
  - Relative boxes take up space where they would have been
- Moved element has same size as if it hadn't been moved
  - Hence specify only one of left/right and top/bottom
    - E.g. if you specify left and right this could change the width of the element, which is not allowed, hence one of left/right will be ignored

See

- http://www.w3.org/TR/CSS21/visuren.html#relative-positioning

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

Margin - Positive and Negative Values

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

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

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

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

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

  - Effectively moves the element 20 pixels to the left

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

Overflow, Min & Max dimensions

- The **`width`** and **`height`** of an object can be constrained
  - With **`min-height`**/**`min-width`** and **`max-height`**/**`max-width`**
  - Once set an element will never grow/shrink beyond these values
- The element is now smaller than the content it display
  - What happens to this content can be controlled with the **`overflow`**
  - Can be set to:
    - **`auto`**
    - **`visible`**
    - **`hidden`**
  - CSS3 allows overflow control on a specific axis **`overflow-x`**/**`y`**
  - In CSS3 we also have the **`hidden`** property

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

Quick Lab 9 – Positioning Elements

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

Lists

CSS FUNDAMENTALS

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`**

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

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

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`**

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>
```

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

@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%
  }
}
```

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 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 Properties (3)

- Shorthand **`animation`** can be used to specify all properties:
- Order:
  - **`duration | timing-function |  delay | iteration-count | direction | fill-mode | play-state | name`**

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

  - 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`** definition

PLAY

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

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`

Transition Properties (2)

- Shorthand **`transition`** can be used to specify all properties:
- Order:
  - **`property | duration | timing-function | delay`**

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

  - 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

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;

}
```

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

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

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

Translations

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

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

```
translateX(tx)
```

```
translateY(ty)
```

- Example:

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

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