Comp 324/424 - Client-side Web Design

Spring Semester 2019 - Week 4

Dr Nick Hayward

CSS Basics - intro

- CSS allows us to define stylistic characteristics for our HTML
  - *helps us define how our HTML is displayed and rendered*
  - *colours used, font sizes, borders, padding, margins, links...*

- CSS can be stored
  - *in external files*
  - *added to a `<style>` element in the `<head>`*
  - *or embedded as inline styles per element*

- CSS not intended as a replacement for encoding semantic and stylistic characteristics with elements

CSS Basics - stylesheet

- add a link to our CSS stylesheet in the `<head>` element

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

- change will replicate throughout our site wherever the stylesheet is referenced

CSS Basics - `<style>` element

- embed the CSS directly within the `<head>` section of our HTML page

- embed using the `<style>` element

- then simply add standard CSS within this element

- limitations include lack of abstraction for site usage and maintenance
  - *styles limited to a single page...*

```
<style type="text/css">
body {
   color: #000;
}
</style>
```

CSS Basics - inline

- embed styles per element using **inline** styles
  - *limitations and detractors for this style of CSS*
  - *helped by the growth and popularity of React...*

e.g.

```
<!-- with styles -->
<p style="color:#cd0603">a trip to Luxor</p>
<!-- without styles -->
<p>a trip to Karnak</p>
```

CSS Basics - pros

Pros

- inherent option and ability to abstract styles from content

- isolating design styles and aesthetics from semantic markup and content

- cross-platform support offered for many aspects of CSS
  - *CSS allows us to style once, and apply in different browsers*
  - *a few caveats remain...*

- various CSS frameworks available

- support many different categories of device
  - *mobile, screen readers, print, TVs...*

- accessibility features

CSS Basics - cons

Cons

- still experience issues as designers with rendering quirks for certain styles
  - *border styles, wrapping, padding, margins...*

- everything is global
  - *CSS matches required selectors against the whole DOM*
  - *naming strategies can be awkward and difficult to maintain*

- CSS can become a mess very quickly
  - *we tend to add to CSS instead of deleting*
  - *can grow very large, very quickly...*

CSS Basics - intro to syntax

- simple, initial concepts for CSS syntax

- follows a defined syntax pattern, e.g.

- selector
  - *e.g. body or p*

- declaration
  - *property and value pairing*

```css
body {
  color: black;
  font-family: "Times New Roman", Georgia, Serif;
  }
```

- body is the selector, `color` is the property, and `black` is the value.

CSS Basics - rulesets

- a CSS file is a group of rules for styling our HTML documents

- rules form **rulesets**, which can be applied to elements within the DOM

- rulesets consist of the following,
  - *a selector - p*
  - *an opening brace - {*
  - *a set of rules - color: blue*
  - *a closing brace - }*

- for example,

```css
body {
  width: 900px;
  color: #444;
  font-family: "Times New Roman", Georgia, Serif;
  }
```

- HTML Colour Picker
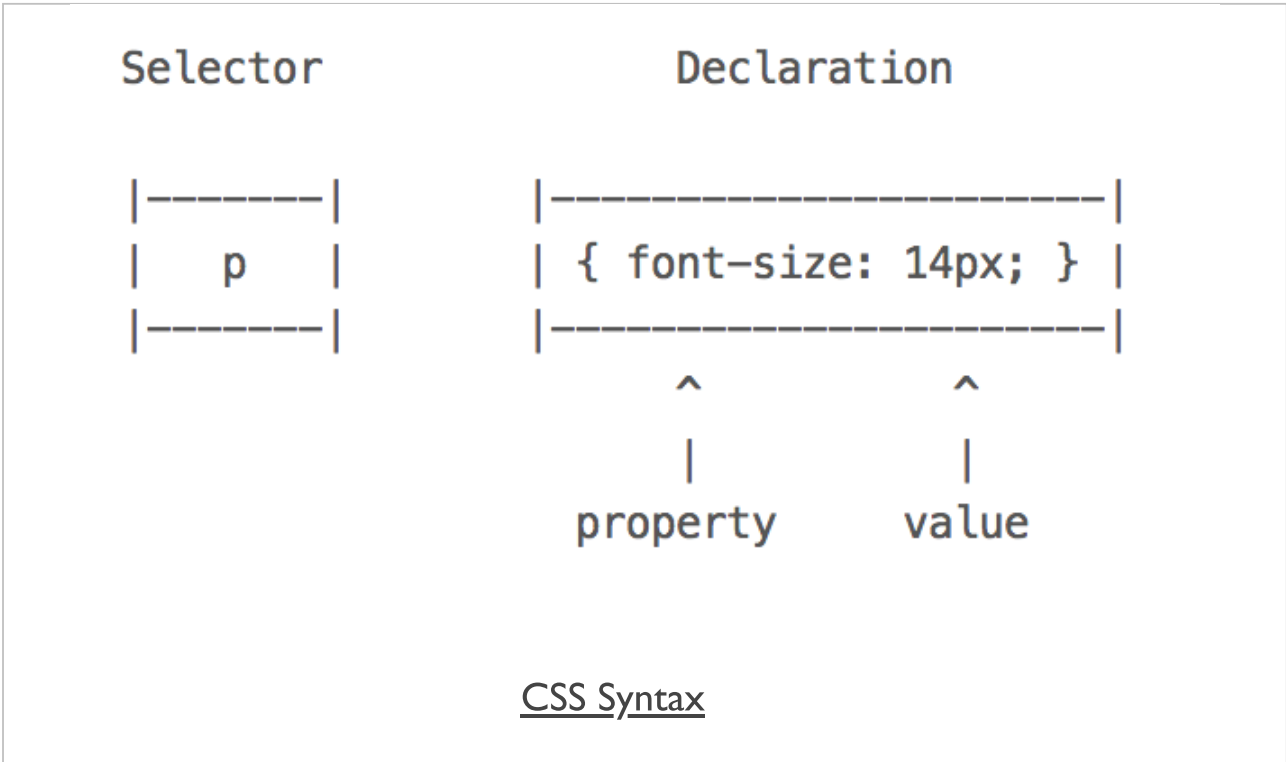
CSS Basics - comments

- add comments to help describe the selector and its properties,

```css
/* 'color' can be set to a named value, HEX value (e.g. #444) &c. */
p {
  color: blue;
  font-size: 14px;
  }
```

- comments can be added before the selector or within the braces
- Demo - CSS Basics

Image - CSS Syntax

```
Selector                Declaration

|————————|              |————————————————————|
|   p    |              | { font-size: 14px; } |
|————————|              |————————————————————|
                             ^              ^
                             |              |
                          property        value


                      CSS Syntax
```

CSS Basics - display

- display HTML elements in one of two ways
  - *inline - e.g. <a> or <span>*
  - *displays content on the same line*

```
<div class="content">
  <p>
    <a href="...">Philae</a> is a <span>Ptolemaic</span> era temple in Egy
  </p>
</div>
```

- more common to display elements as `block-level` instead of `inline` elements
- element's content rendered on a new line outside flow of content
- a few sample block elements include,
  - *<article>, <div>, <figure>, <main>, <nav>, <p>, <section>...*
- *block-level* is not technically defined for new elements in HTML5
- Demo - CSS Basics - Add a Class

CSS Basics - inline elements

Current inline elements include, for example:

- b | big | i | small
- abbr | acronym | cite | dfn | em | strong | var
- a | br | img | map | script | span | sub | sup
- button | input | label | select | textarea
- ...

Source - MDN - Inline Elements

n.b. not all inline elements supported in HTML5

CSS Basics - block-level elements

Current block-level elements include:

- address | article | aside | blockquote | canvas | div
- fieldset | figure | figcaption | footer | form
- h1 | h2 | h3 | h4 | h5 | h6
- header | hgroup | hr | main | nav
- ol | output | p | pre | section | table | tfoot | ul | video
- ...

Source - MDN - Block-level Elements

n.b. *block-level* is not technically defined for new elements in HTML5

CSS Basics - HTML5 content categories - part 1

- **block-level** is not technically defined for new elements in HTML5
- now have a slightly more complex model called **content categories**
- includes three primary types of content categories

These include,

- **main content categories** - describe common content rules shared by many elements
- **form-related content categories** - describe content rules common to form-related elements
- **specific content categories** - describe rare categories shared by only a small number of elements, often in a specific context

CSS Basics - HTML5 content categories - part 2

- **Metadata content** - modify presentation or behaviour of document, setup links, convey additional info...
  - *<base>, <command>, <link>, <meta>, <noscript>, <script>, <style>, <title>*

- **Flow content** - typically contain text or embedded content
  - *<a>, <article>, <canvas>, <figure>, <footer>, <header>, <main>...*

- **Sectioning content** - create a section in current outline to define scope of <header> elements, <footer> elements, and *heading* content
  - *<article>, <aside>, <nav>, <section>*

- **Heading content** - defines title of a section, both explicit and implicit sectioning
  - *<h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <hgroup>*

Source - MDN Content Categories
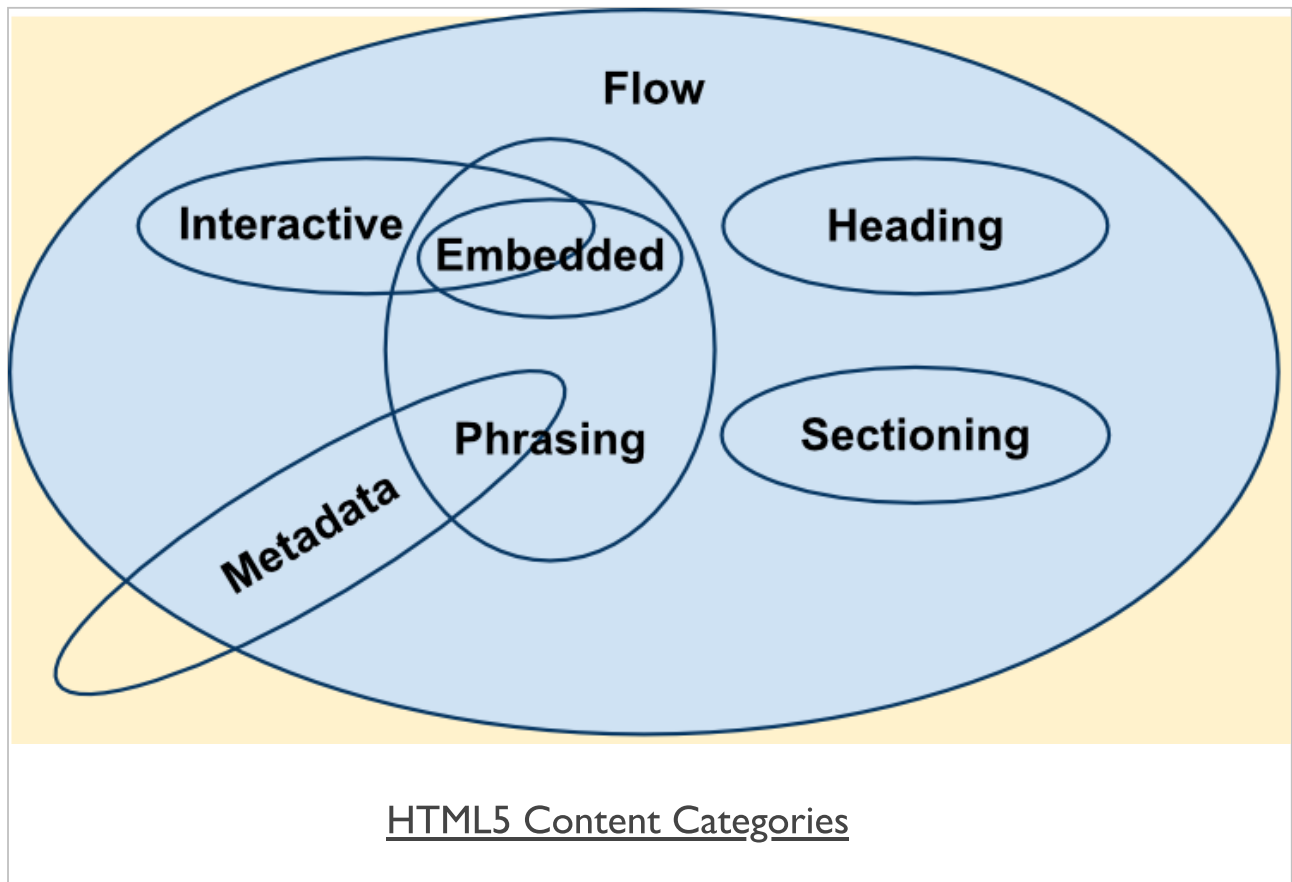
CSS Basics - HTML5 content categories - part 3

- **Phrasing content** - defines the text and the mark-up it contains
  - *<audio>, <canvas>, <code>, <img>, <label>, <script>, <video>...*
  - *other elements can belong to this category if certain conditions are met. e.g. <a>*

- **Embedded content** - imports or inserts resource or content from another mark-up language or namespace
  - *<audio>, <canvas>, <embed>, <iframe>, <img>, <math>, <object>, <svg>, <video>*

- **Interactive content** - includes elements that are specifically designed for user interaction
  - *<a>, <button>, <details>, <embed>, <iframe>, <keygen>, <label>, <select>, <textarea>*
  - *additional elements, available under specific conditions, include*
  - *<audio>, <img>, <input>, <menu>, <object>, <video>*

- **Form-associated content** - elements contained by a form parent element
  - *<button>, <input>, <label>, <select>, <textarea>...*
  - *there are also several sub-categories, including listed, labelable, submittable, resettable*

Source - MDN Content Categories

Image - HTML5 Content Categories

HTML5 Content Categories

Source - MDN - Content Categories

CSS Basics - box model - part 1

- consideration of the CSS box model

- a document's attempt to represent each element as a rectangular box

- boxes and properties determined by browser rendering engine

- browser calculates size, properties, and position of these required boxes

- properties can include, for example,
  - *colour, background features, borders, width, height...*

- box model designed to describe an element's required space and content

- each box has a series of edges,
  - **margin** *edge*
  - **border** *edge*
  - **padding** *edge*
  - **content** *edge*

CSS Basics - box model - part 2

Content

- box's **content area** describes element's actual content
- properties can include `color, background, img...`
  - *apply inside the **content** edge*
- dimensions include **content width** and **content-height**
- content size properties (assuming that the `box-sizing` property remains default) include,
  - *width, min-width, max-width, height, min-height, max-height*

Demo - CSS Box Model

- Demo - CSS Box Model

CSS Basics - box model - part 3

Padding

- box's **padding area** includes the extent of the padding to the surrounding border

- background, colour etc properties for a content area extend into the padding
  - *we often consider the padding as extending the content*

- padding itself is located in the box's **padding edge**

- dimensions are the width and height of the **padding-box**.

- control space between padding and content edge using the following properties,
  - `padding-top, padding-right, padding-bottom, padding-left`
  - `padding` *(sizes calculated clock-wise)*

Demo - CSS Box Model - Padding

- JSFiddle - CSS Box Model

CSS Basics - box model - part 4

Border

- **border area** extends **padding area** to area containing the borders

- it becomes the area inside the **border edge**

- define its dimensions as the width and height of the **border-box**

- calculated area depends upon the width of the `border` we set in the CSS

- set size of our border using the following properties in CSS,
  - *border-width*
  - *border*

Demo - CSS Box Model - Border

- JSFiddle - CSS Box Model

CSS Basics - box model - part 5

Margin

- **margin area** can extend this border area with an empty area
  - *useful to create a defined separation of one element from its neighbours*

- dimensions of area defined as width and height of the **margin-box**

- control size of our margin area using the following properties,
  - `margin-top`, `margin-right`, `margin-bottom`, `margin-left`
  - `margin` *(sizes calculated clock-wise)*

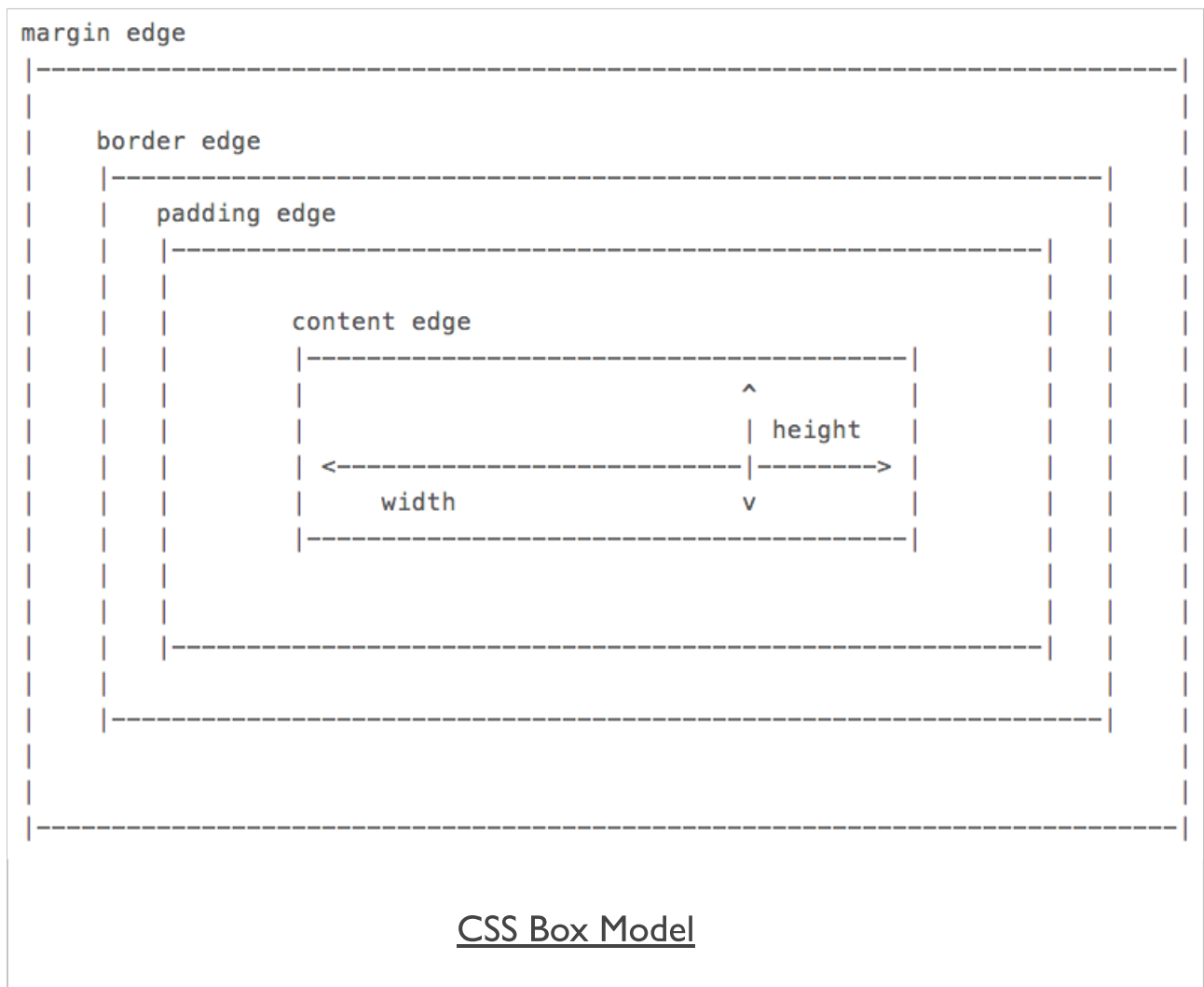# Demo - CSS Box Model - Margin

- JSFiddle - CSS Box Model

Demo - CSS Box Model

- Demo - CSS Box Model

Image - CSS Box Model

```
margin edge
 |----------------------------------------------------------------|
 |                                                                |
 |    border edge                                                 |
 |    |------------------------------------------------------|    |
 |    |    padding edge                                       |    |
 |    |    |--------------------------------------------|     |    |
 |    |    |                                            |     |    |
 |    |    |    content edge                            |     |    |
 |    |    |    |-----------------------------------|   |     |    |
 |    |    |    |                             ^     |   |     |    |
 |    |    |    |                             | height    |   |     |    |
 |    |    |    | <---------------------------|-------> | |     |    |
 |    |    |    |    width                    v       |   |     |    |
 |    |    |    |-----------------------------------|   |     |    |
 |    |    |                                            |     |    |
 |    |    |                                            |     |    |
 |    |    |--------------------------------------------|     |    |
 |    |                                                       |    |
 |    |------------------------------------------------------|    |
 |                                                                |
 |                                                                |
 |----------------------------------------------------------------|
```

CSS Box Model

Source - MDN - CSS Box Model

Demo - CSS Box Model - Interactive

- interactive Box Model

CSS Box Model - structure and layout

fun exercise

Choose one of the following app examples,

- magazine or news reader
  - *e.g. a local newspaper, or perhaps a news aggregator*

- social media aggregator
  - *collect and display updates and news from various social media APIs*

- gaming portal for a community
  - *collect latest scores, news, comments, photos &c. for a chosen game*

Then, consider the following

- use of the box model to layout your example pages
  - *where is it being used?*
  - *why is it being used for a given part of the UI?*

- rendering of box model in the main content
  - *i.e. box model updates due to changes in content*

- which parts of the UI will not benefit from box model?

CSS Basics - selectors

- **selectors** are a crucial part of working with CSS, JS...

- basic selectors such as

```css
p {
  color: #444;
}
```

- above ruleset adds basic styling to our paragraphs
  - *sets the text colour to HEX value 444*

- simple and easy to apply
  - *applies the same properties and values to all paragraphs*

- specificity requires classes, pseudoclasses...

CSS Basics - classes

- add a **class** attribute to an element, such as a `<p>`
  - *can help us differentiate elements*

- also add a **class** to any DOM element
  - *e.g. add different classes to multiple <p> elements*

```html
<p class="p1">paragraph one...</p>
<p class="p2">paragraph two...</p>
```

- we can now select our paragraphs by class name within the DOM

- then apply a **ruleset** for each class

- style this class for a specific element

```css
p.p1 {
   color: #444;
}
```

- style all elements with the class p1, and not just `<p>` elements

```css
.p1 {
   color: #444;
}
```

CSS Basics - pseudoclasses

- add a class to links or anchors, styling all links with the same ruleset

- we might also want to add specific styles for different link states

- styling links with a different colour
  - *e.g. whether a link has already been used or not*

```css
a {
  color: blue;
  }

a:visited {
  color: red;
  }
```

- `visited` is a CSS **pseudoclass** applied to the <a> element

- browser implicitly adds this pseudoclass for us, we add style

```css
a:hover {
  color: black;
  text-decoration: underline;
}
```

- pseudoclass for link element, <a>, `hover`

CSS Basics - complex selector - part 1

- our DOM will often become more complicated and detailed

- depth and complexity will require more complicated selectors as well

- lists and their list items are a good example

```
<ul>
   <li>unordered first</li>
   <li>unordered second</li>
   <li>unordered third</li>
</ul>
<ol>
   <li>ordered first</li>
   <li>ordered second</li>
   <li>ordered third</li>
</ol>
```

- two lists, one unordered and the other ordered

- style each list, and the list items using rulesets

```
ul {
   border: 1px solid green;
}
ol {
   border: 1px solid blue;
}
```

Demo - Complex Selectors - Part 1

- Demo - Complex Selectors Part 1

CSS Basics - complex selector - part 2

- add a ruleset for the list items, `<li>`

- applying the same style properties to both types of lists

- more specific to apply a ruleset to each list item for the different lists

```css
ul li {
   color: blue;
}
ol li {
   color: red;
}
```

- also be useful to set the background for specific list items in each list

```css
li:first-child {
   background: #bbb;
}
```

- pseudoclass of `nth-child` to specify a style for the second, fourth etc child in the list

```css
li:nth-child(2) {
   background: #ddd;
}
```

Demo - Complex Selectors - Part 2

- Demo - Complex Selectors Part 2

CSS Basics - complex selector - part 3

- style odd and even list items to create a useful alternating pattern

```css
li:nth-child(odd) {
  background: #bbb;
}
li:nth-child(even) {
  background: #ddd;
}
```

- select only certain list items, or rows in a table etc
  - *e.g. every fourth list item, starting at the first one*

```css
li:nth-child(4n+1) {
  background: green;
}
```

- for **even** and **odd** children we're using the above with convenient shorthand

- other examples include
  - *last-child*
  - *nth-last-child()*
  - *many others...*

Demo - CSS Complex Selectors - Part 3

- Demo - Complex Selectors Part 3

CSS Basics - cascading rules - part 1

- CSS, or cascading style sheets, employs a set of **cascading** rules

- rules applied by each browser as a ruleset conflict arises
  - *e.g. issue of **specificity***

```css
p {
   color: blue;
   }
p.p1 {
   color: red;
   }
```

- the more specific rule, the class, will take precedence

- issue of possible duplication in rulesets

```css
h3 {
   color: black;
}

h3 {
   color: blue;
}
```

- **cascading** rules state the later ruleset will be the one applied
  - *blue heading instead of black...*

CSS Basics - cascading rules - part 2

- simple styling and rulesets can quickly become compounded and complicated

- different styles, in different places, can interact in complex ways

- a powerful feature of CSS
  - *can also create issues with logic, maintenance, and design*

- three primary sources of style information that form this cascade

  1. default styles applied by the browser for a given markup language
     - *e.g. colours for links, size of headings...*

  2. styles specific to the current user of the document
     - *often affected by browser settings, device, mode...*

  3. styles linked to the document by the designer
     - *external file, embedded, and as inline styles per element*

CSS Basics - cascading rules - part 3

- basic cascading nature creates the following pattern
  - *browser's style will be default*
  - *user's style will modify the browser's default style*
  - *styles of the document's designer modify the styles further*

CSS Basics - inheritance

- CSS includes inheritance for its styles

- descendants will inherit properties from their ancestors

- style an element
  - *descendants of that element within the DOM inherit that style*

```css
body {
  background: blue;
}
p {
  color: white;
}
```

- p is a descendant of body in the DOM
  - *inherits background colour of the body*

- this characteristic of CSS is an important feature
  - *helps to reduce redundancy and repetition of styles*

- useful to maintain outline of document's DOM structure

- most styles follow this pattern but not all

- margin, padding, and border rules for block-level elements **not inherited**

CSS Basics - fonts - part 1

- fonts can be set for the body or within an element's specific ruleset

- we need to specify our font-family,

```css
body {
font-family: "Times New Roman", Georgia, Serif;
}
```

- value for the font-family property specifies preferred and fall-back fonts
  - *Times New Roman, then the browser will try Georgia and Serif*
  - *" " - quotation marks for names with spaces...*

n.b. " " added due to CSS validator requesting this standard - it's believed to be a legacy error with the validator...

CSS Basics - fonts - part 2

- useful to be able to modify the size of our fonts as well

```css
body {
   font-size: 100%;
}
h3 {
   font-size: x-large;
}
p {
   font-size: larger;
}
p.p1 {
   font-size: 1.1em;
}
```

- set base font size to 100% of font size for a user's web browser

- scale our other fonts relative to this base size
  - *CSS absolute size values, such as* `x-large`
  - *font sizes relative to the current context, such as* `larger`
  - *em are meta-units, which represent a multiplier on the current font-size*
  - *relative to current element for required font size*
  - *1.5em of 12px is effective 18px*

- em font-size scales according to the base font size
  - *modify base font-size, em sizes adjust*

- try different examples at
  - *W3 Schools - font-size*

Demo - CSS Fonts

- Demo - CSS Fonts
- JSFiddle - CSS Fonts

CSS Basics - fonts - part 3

- `rem` unit for font sizes
- size calculated against root of document

```css
body {
  font-size: 100%;
}
p {
  font-size: 1.5rem;
}
```

- element font-size will be `root size * rem size`
  - *e.g. body font-size is currently 16px*
  - *rem will be* `16 * 1.5`

CSS Basics - custom fonts

- using fonts and CSS has traditionally been a limiting experience

- reliant upon the installed fonts on a user's local machine

- JavaScript embedding was an old, slow option for custom fonts

- web fonts are a lot easier

- Google Fonts
  - *from the font options, select*
  - *required fonts*
  - *add a `<link>` reference for the font to our HTML document*
  - *then specify the fonts in our CSS*

```
font-family: 'Roboto';
```

Demo - CSS Custom Fonts

- Demo - CSS Custom Fonts
- JSFiddle - CSS Custom Fonts

CSS Basics - reset options

- to help us reduce browser defaults, we can use a CSS reset

- reset allows us to start from scratch

- customise aspects of the rendering of our HTML documents in browsers

- often considered a rather controversial option

- considered controversial for the following primary reasons
  - *accessibility*
  - *performance*
  - *redundancy*

- use resets with care

- notable example of resets is Eric Meyer
  - *discussed reset option in May 2007 blog post*

- resets often part of CSS frameworks...

Browser default styles are used for

- `<h1>, <h3>, and <p>`
- Demo - CSS Reset Before

Browser resets are implemented using the Eric Meyer stylesheet.

- Demo - CSS Reset After

CSS - a return to inline styles

- *inline* styles are once more gaining in popularity
  - *helped by the rise of React &c.*

- for certain web applications they are now an option
  - *allow us to dynamically maintain and update our styles*

- their implementation is not the same as simply embedding styles in HTML
  - *dynamically generated*
  - *can be removed and updated*
  - *can form part of our maintenance of the underlying DOM*

- inherent benefits include
  - *no cascade*
  - *built using JavaScript*
  - *styles are dynamic*

CSS - against inline styles

- CSS is designed for styling
  - *this is the extreme end of the scale - in effect, styling is only done with CSS*

- abstraction is a key part of CSS
  - *by separating out concerns, i.e. CSS for styling, our sites are easier to maintain*

- *inline* styles are too specific
  - *again, abstraction is the key here*

- some styling and states are easier to represent using CSS
  - *psuedoclasses etc, media queries...*

- CSS can add, remove, modify classes
  - *dynamically update selectors using classes*

Demos

- CSS Basics
- CSS Basics - Add a Class
- CSS - Box Model
- CSS - Complex Selectors Part 1
- CSS - Complex Selectors Part 2
- CSS - Complex Selectors Part 3
- CSS - Interactive Box Model

- *CSS - Fonts*
- *CSS Fonts*
- *CSS Custom Fonts*
- *CSS Reset Before*
- *CSS Reset After*
- *JSFiddle tests - CSS*
- *CSS Box Model*
- *CSS Box Model Padding*
- *CSS Fonts*
- *CSS Custom Fonts*

Resources

- HTML Colour Picker
- MDN - Block-level Elements
- MDN - Content Categories
- MDN - CSS Box Model
- MDN - CSS Selectors
- MDN - Inline Elements
- Google Web Fonts