

Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 9 - Friday

Dr Nick Hayward

CSS Basics - complex selector - part I

- 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 I

- Demo - Complex Selectors Part I

CSS Basics - complex selector - part 2

- add a ruleset for the list items, ``
- applying the same style properties to both types of lists
- more specific to apply a ruleset to each list item for the different lists

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

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

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

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

```
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

```
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*

```
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 I

- 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**

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

- the more specific rule, the class, will take precedence
- issue of possible duplication in rulesets

```
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*

```
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 I

- fonts can be set for the `body` or within an element's specific ruleset
- we need to specify our font-family,

```
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

```
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

```
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*

```
p {  
  font-family: 'Roboto';  
}
```


Demo - CSS Custom Fonts

- [Demo - CSS Custom Fonts](#)
- [JSFiddle - CSS Custom Fonts](#)

References

- MDN
 - *CSS documentation*
 - *CSS Selectors*
- W3Schools
 - *CSS*
 - *CSS Box Model*
 - *CSS - Selectors Reference*