Comp 388/424 - Client-side Web Design

Spring Semester 2016 - Week 4

Dr Nick Hayward

Contents

Outline

Up to DEV week

- JS intro, basics, and fundamentals
- JS advanced, JQuery, APIs...
- basic JS tools, workflows, and methods
- testing, initial deployment
  - *pulling it all together*

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
  - *eg: 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
     - *eg: 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*

- 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 1

- Fonts can be set for the body or within an element's specific ruleset
- we need to do 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*

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*
  - *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 1 - CSS Fonts
- JSFiddle - CSS Fonts

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
  - *pick and choose our custom fonts by selecting Quick-use*
  - *from the options, select*
  - *required character sets*
  - *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 2 - CSS Custom Fonts
- JSFiddle - CSS Custom Fonts

CSS Basics - reset options

- help us reduce browser defaults, we can use a CSS reset
- often considered a rather controversial option
- reset allows us to start from scratch
- customise aspects of the rendering of our HTML documents in browsers
- 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...

Demo - CSS Reset - Before

Browser default styles are used for

- <h1>, <h3>, and <p>
- Demo 3 - CSS Reset Before

Demo - CSS Reset - After

Browser resets are implemented using the Eric Meyer stylesheet.

- Demo 4 - CSS Reset After

CSS - a return to inline styles

- *inline* styles are oncemore gaining in popularity
  - *helped by the rise of React*
- 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, ie: 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*

JS Intro

- JavaScript (JS) a core technology for client-side design and development
- now being used as a powerful technology to help us
  - *rapidly prototype and develop web, mobile, and desktop apps*
- libraries such as JQuery, React, AngularJS, and Node.js
- helps develop cross-platform apps
  - *Apache Cordova*
  - *Electron*

JS Basics - operators

- operators allow us to perform
  - *mathematical calculations*
  - *assign one thing to another*
  - *compare and contrast...*
- simple * operator, we can perform multiplication

```
2 * 4
```

- we can add, subtract, and divide numbers as required
- mix mathematical with simple assignment

```
a = 4;
b = a + 2;
```

JS Basics - some common operators - part 1

Assignment

- = eg: `a = 4`

Comparison

- <, > <=, >=
- eg: `a <= b`

Compound assignment

- +=, -=, *=, /=
- compound operators combine a mathematical operation with assignment
- eg: `a += 4`

Equality

operator	description
==	loose equals
===	strict equals
!=	loose not equals
!==	strict not equals

- eg: `a != b`

JS Basics - some common operators - part 2

Increment/Decrement

- increment or decrement an existing value by 1
  - *`++`, `--`, ``*
  - *eg: `a++` is equal to `a = a + 1`*

Logical

- used to express compound conditionals - **and**, **or**
  - *`&&`, `||`*
  - *eg: `a || b`*

Mathematical

- +, –, *, /
  - *eg: `a * 4` or `a / 4`*

Object property access

- properties in objects are specific named locations for holding values and data
- effectively, values within values
  - *`.`*
  - *eg: `a.b` means object `a` with a property of `b`*

JS Basics - values and types

- able to express different representations of values
  - *often based upon need or intention*
  - *known as* ***types***
- JS has built-in types
  - *allow us to represent* ***primitive*** *values*
  - *eg:* ***numbers***, ***strings***, ***booleans***
- such values in the source code are simply known as **literals**
- **literals** can be represented as follows,
  - *string literals use double or single quotes eg:* `"some text"` *or* `'some more text'`
  - *numbers and booleans are represented without being escaped eg:* `49`, `true;`
- also consider arrays, objects, functions...

JS Basics - type conversion

- option and ability to convert types in JS
  - *in effect,* ***coerce*** *our values and types from one type to another*
- convert a number, or coerce it, to a string
- built-in JS function, `Number()`, is an explicit coercion
  - *explicit coercion, convert any type to a number type*
- implicit coercion, JS will often perform as part of a comparison

```
"49" == 49
```

- JS implicitly coerces left string to a matching number
  - *then performs the comparison*
- often considered bad practice
  - *convert first, and then compare*
- implicit coercion still follows rules
  - *can be very useful*

JS Basics - variables - part I

- **symbolic** container for values and data
- applications use containers to keep track and update values
- use a **variable** as a container for such values and data
  - *allow values to vary over time*
- JS emphasises types for values, does not enforce on the variable
  - ***weak typing*** *or* ***dynamic typing***
  - *JS permits a variable to hold a value of any type*
- often a benefit of the language
- a quick way to maintain flexibility in design and development

JS Basics - variables - part 2

- declare a variable using the keyword `var`
- declaration does not include **type** information

```
var a = 49;
//double var a value
var a = a * 2;
//coerce var a to string
var a = String(a);
//output string value to console
console.log(a);
```

- `var a` maintains a running total of the value of `a`
- keeps record of changes, effectively **state** of the value
- **state** is keeping track of changes to any values in the application

JS Basics - variables - part 3

- use variables in JS to enable central, common references to our values and data
- better known in most languages simply as **constants**
- allow us to define and declare a variable with a value
  - *not intended to change throughout the application*
- **constants** are often declared together
- form a store for values abstracted for use throughout an app
- JS normally defines constants using uppercase letters,

```
var NAME = "Philae";
```

- ECMAScript 6, ES6, uses the keyword `const` instead of `var`

```
const TEMPLE_NAME = "Philae";
```

- benefits of abstraction, ensuring value is not accidentally changed

JS Basics - comments

- JS permits comments in the code
- two different implementations

single line

```
//single line comment
var a = 49;
```

multi-line

```
/* this comment has more to say
hence the need for more lines... */
var b = "forty nine";
```

CSS - test and try out

- JSFiddle - CSS Custom Fonts
- JSFiddle - CSS Fonts

Demos

- Demo 1 - CSS Fonts
- Demo 2 - CSS Custom Fonts
- Demo 3 - CSS Reset Before
- Demo 4 - CSS Reset After

References

CSS & Typography

- Eric Meyer - reset CSS
- MDN - CSS
- Cascading and inheritance
- Perishable Press - Barebones Web Templates
- Typography - EM units
- The Unicode Consortium
- Unicode Information
- Unicode examples
- W3 CSS
- W3 Schools - CSS
- W3 Schools - font-size

JavaScript & Libraries

- AngularJS
- Apache Cordova
- Electron
- JQuery
- MDN - JS
  - *MDN - JS Grammar and Types*
  - *MDN - JS Objects*
- Node.js
- React