Comp 324/424 - Client-side Web Design

Spring Semester 2024 Week 4

Dr Nick Hayward

Project outline & mockup assessment

Course total = 15 credits

- begin development and design of a web application
  - built from scratch
    - HTML5, CSS…
  - builds upon examples, technology outlined during first part of semester
  - purpose, scope &c. is group's choice
  - **NO** blogs, to-do lists, note-taking…
  - presentation should include initial designs, mockups, and any current HTML5 and CSS

Project outline & mockup assessment

Assessment will include the following:

- brief presentation or demonstration of current project work
  - ~ 10 minutes per group
  - analysis of work conducted so far
  - presentation and demonstration…
    - outline initial project idea and concept
    - outline current state of web app concept and design
    - show mockups and designs
    - …
  - due Monday 12th February 2024 @ 4.15pm

Building a web app - sample outline of underlying structure

- apps developed using a full JavaScript stack
- using and incorporating JS into each part of app's development
  - UI front-end
  - app server and management
  - data store and management
- Technologies will include
  - front-end: HTML5, CSS, JS…
  - app server: Node.js, Express…
  - data store: MongoDB, Redis, Mongoose…
- Data format is JSON

Image - building a web app - sample outline

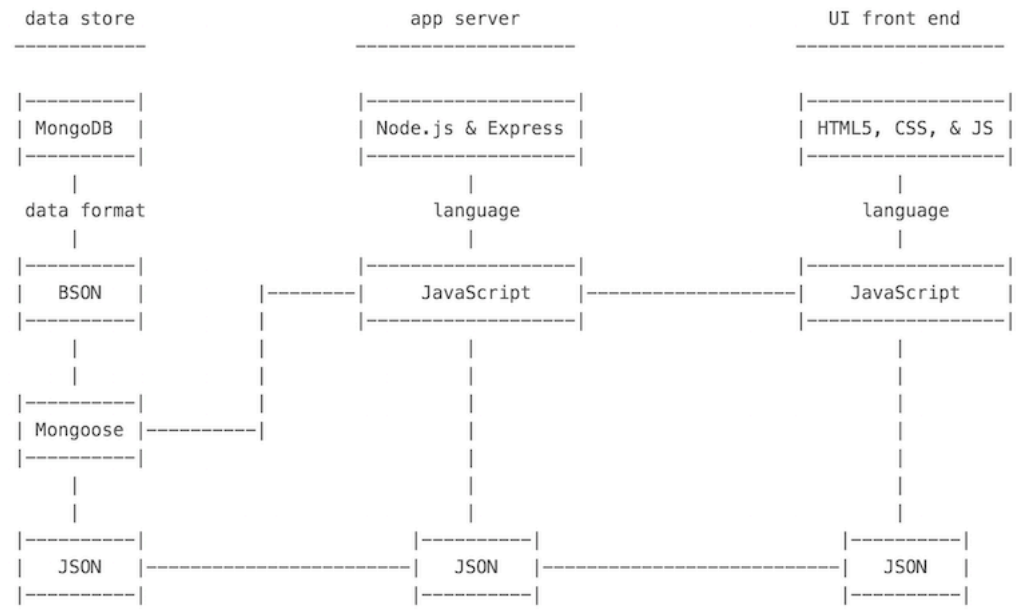

Figure 1: JS full-stack outline

n.b. I've explicitly omitted any arrows for flow within this diagram. This is something we'll return to as we start to work with Node.js, Mongoose, MongoDB &c.

HTML5, CSS, & JS - example - part 1

Structure

- combine HTML5, CSS, and JavaScript, to create an example application
- outline of our project's basic directory structure

```
.
|- assets
|  |- images //logos, site/app banners - useful images for site's design
|  |- scripts //js files
|  |- styles  //css files
|- docs
|  |- json //any .json files
|  |- txt //any .txt files
|  |- xml //any .xml files
|- media
|  |- audio //local audio files for embedding & streaming
|  |- images //site images, photos
|  |- video //local video files for embedding & streaming
|- index.html
```

- each of the above directories can, of course, contain many additional sub-directories
  - `|- images` may contain sub-directories for albums, galleries…
  - `|- xml` may contain sub-directories for further categorisation..

- and so on...

Video - Navigation Map

planning a site design Source: Designing a website - YouTube

HTML5, CSS, & JS - example - part 2

index.html

- initial HTML structure for app

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>travel notes - v0.1</title>
    <meta name="description" content="information on travel destinations">
    <meta name="author" content="ancientlives">
    <!-- css styles... -->
    <link rel="stylesheet" type="text/css" href="assets/styles/style.css">
  </head>
  <body>
    ...
    <!-- js scripts... -->
    <script type="text/javascript" src="assets/scripts/travel.js"></script>
  </body>
</html>
```

- JS files at foot of body
  - hierarchical rendering of page by browser - top to bottom
  - JS will now be one of the last things to load
  - JS files often large, slow to load
  - helps page load faster...

HTML5, CSS, & JS - example - part 3

index.html - body

- add initial app structure

```
<body>
  <!-- document header -->
  <header>
    <h3>travel notes</h3>
    <p>record notes from various cities and placed visited...</p>
  </header>
  <!-- document main -->
  <main>
    <!-- note input -->
    <section class="note-input">
    </section>
    <!-- note output -->
```

```
    <section class="note-output">
    </section>
  </main>
  <!-- document footer -->
  <footer>
    <p>app's copyright information, additional links...</p>
  </footer>
  <!-- js scripts... -->
  <script type="text/javascript" src="assets/scripts/travel.js"></script>
</body>
```

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

```
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,

```
  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,

```
/* '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

HTML5, CSS, & JS - example - part 4

style.css

- initial styles for app

```
body {
  width: 850px;
  margin: auto;
  background: #fff;
  font-size: 16px;
  font-family: "Times New Roman", Georgia, Serif;
}
h3 {
  font-size: 1.75em;
}
header {
  border-bottom: 1px solid #dedede;
```

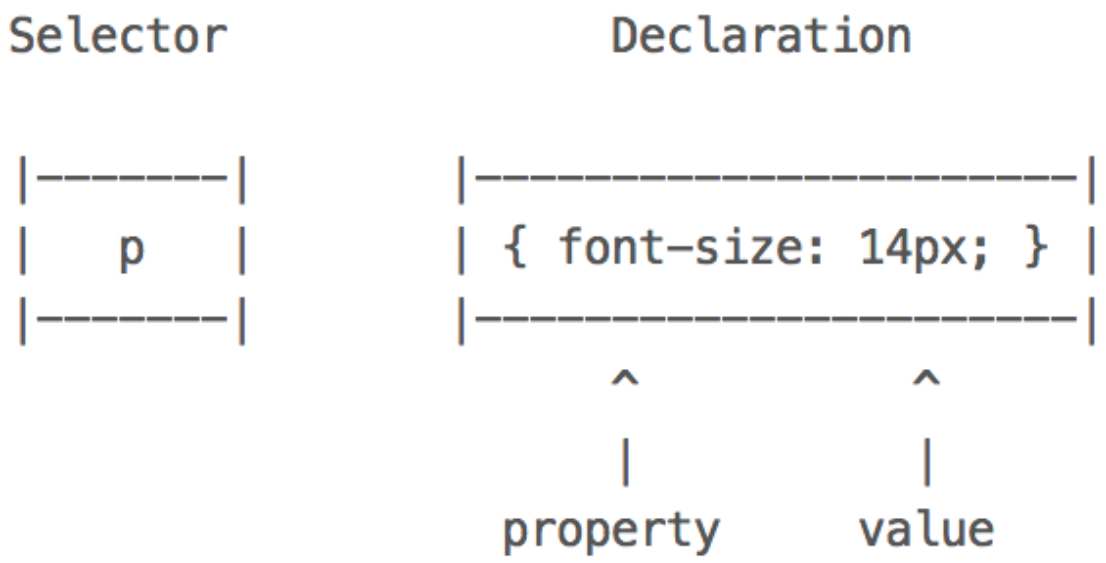

Figure 2: CSS Syntax

```
}
header p {
  font-size: 1.25em;
  font-style: italic;
}
footer p {
  font-size: 0.8em;
}
```

Video - CSS and Fonts

Typography considerations - part 1 Typography - up to 2:13

Source - Typography - YouTube

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,

```
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

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

Demo - CSS Custom Fonts

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

Video - CSS and Fonts

Typography considerations - part 2 Typography - up to 3:33

Source - Typography - YouTube

HTML5, CSS, & JS - example - part 5

add a note

- app's structure includes three clear semantic divisions of content
  - `<header>` , `<main>` , and `<footer>`
- `<main>` content category - create and add our notes for our application
- allow a user to create a new note
  - enter some brief text, and then set it as a note
- output will simply resemble a heading or brief description for our note
- add HTML element `<input>` to allow a user to enter note text
  - new attributes in HTML5 such as autocomplete, autofocus, required, width…
  - set accompanying

```
<h5>add note</h5>
<input>
```

```
<input type="text" value="add a note...">
```

HTML5, CSS, & JS - example - part 6

tidy up styling

- additional styles to create correct, logical separation of visual elements and content
- add a border to the top of our footer
  - perhaps matching the header in style
- update the box model for the `<main>` element

- add some styling for `<h5>` heading

```
h5 {
  font-size: 1.25em;
  margin: 10px 0 10px 0;
}
main {
  overflow: auto;
  padding: 15px 0 15px 0;
}
footer {
  margin-top: 5px;
  border-top: 1px solid #dedede;
}
```

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 Egypt.
  </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

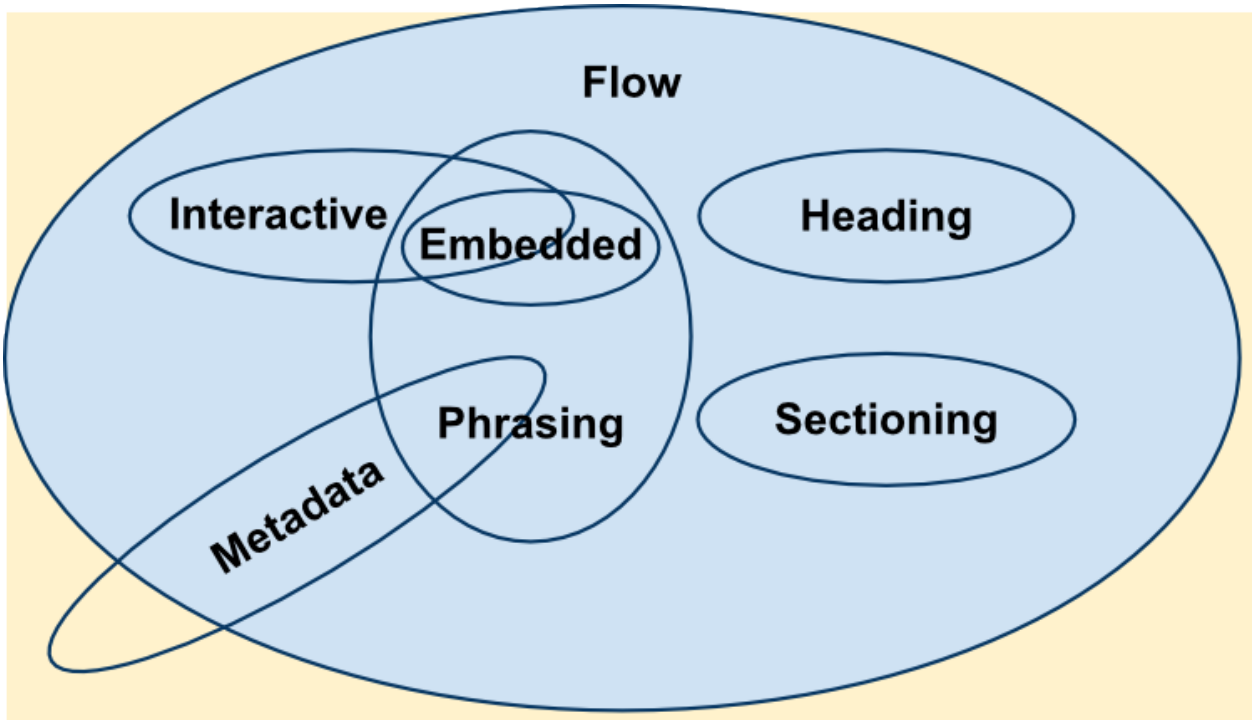

Figure 3: HTML5 Content Categories

Source - MDN - Content Categories

Video - Organising Visual Data

genius of London Tube map TED: The Genius of the London Tube Map Source: Genius of the London Tube Map - YouTube

---

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

```
  <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

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

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

```
  .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

```
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

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

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

Video - Shade and shadow

Grey Square Optical Illusion - Source: YouTube

HTML5, CSS, & JS - example - part 7

```
<input><button>add</button>
```

```
.note-input input {
  width: 40%;
}
.note-input button {
  padding: 2px;
  margin-left: 5px;
  border-radius: 0;
  border: 1px solid #dedede;
  cursor: pointer;
}
```

input update

- also update css for `input` and `button`
- remove button's rounded borders to match style of `input`
- match border for button to basic design aesthetics
- set cursor appropriate for a link style…
- DEMO - travel notes - series 1

Video - Data and Persistency

data and design in UX Source: YouTube

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
- Embedded systems
  - Espruino - http://www.espruino.com/
  - Tessel - https://tessel.io/

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

- `=`
- e.g. `a = 4`

Comparison

- `<` , `>` `<=` , `>=`
- e.g. `a <= b`

Compound assignment

- `+=` , `-=` , `*=` , `/=`
- compound operators are used to combine a mathematical operation with assignment
- same as `result = result + expression`
- e.g. `a += 4`

Equality

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

- e.g. `a != b`

JS Basics - some common operators - part 2

Increment/Decrement

- increment or decrement an existing value by 1
  - `++` , `--`
  - e.g. `a++` is equal to `a = a + 1`

Logical

- used to express compound conditionals - **and**, **or**
  - `&&` , `||`
  - e.g. `a || b`

Mathematical

- `+` , `-` , `*` , `/`
  - e.g. `a * 4` or `a / 4`

Object property access

- properties in objects are specific named locations for holding values and data
- effectively, values within values
  - `.`
  - e.g. `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
  - e.g. **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 e.g. `"some text"` or `'some more text'`
  - *numbers* and *booleans* are represented without being escaped e.g. `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

Demos

CSS

- CSS Basics
- CSS Basics - Add a Class

Travel Notes - series 1

- travel notes - demo 1

Resources

- Google Web Fonts
- HTML Colour Picker
- JS
  - MDN - JS
  - JavaScript - Scope and variables - YouTube
  - JS Info - DOM Nodes
    - MDN - JS Grammar and Types
    - W3 Schools - JS
- MDN Documentation
  - Block-level Elements
  - Content Categories
  - CSS Selectors
  - HTML developer guide