Comp 125 - Visual Information Processing

Spring Semester 2019 - Week 9 - Monday

Dr Nick Hayward

HTML & JS - Random Greeting Generator

demo - basic

- Random Greeting Generator - Basic

HTML & JS - Random Greeting Generator - variant 2

example solution - update JS logic

- abstract JS logic with function `generateGreeting()`
  - *add `greetings` array*
  - *get random `greeting`*
  - *return `greeting` value from function*
  - *accept parameter for `name`*
  - *use `name` with*

```
// FN: greetings generator
function generateGreeting(name) {
    // define random greetings - initial fixed examples...
    let greetings = [
        `Hello ${name}, how are you?`,
        `Bonjour ${name}, ça va? `,
        `Guten tag ${name}, wie geht es ihnen?`,
        `Χαίρετε ${name}, Πώς είσαι;`,
        `Salve ${name}, quid agis?`,
        `Ciao ${name}, come va?`,
        `こんにちは ${name}, お元気ですか?`
    ];
    // pick a random greeting message
  let greeting = greetings[Math.floor(Math.random() * greetings.length)];
    // return greeting message
    return greeting;
}
```

HTML & JS - Random Greeting Generator

demo - update JS logic

- Random Greeting Generator - Better

JS - ES6 template literals

an updated option for concatenation

- concatenate strings, values, variables &c. using **template literals**
  - *new to ES6 (ES2015) JavaScript update*

```
`Hello ${name}, how are you?`
```

- start and end string with a backtick (grave accent in French)

```
`...`
```

- add string

```
`Hello`
```

- then inject variable, value &c. into template literal with string
  - *adds required code for concatenation with string*

```
${name}
```

HTML & JS - Random Greeting Generator - variant 2

example solution - update JS logic

- update event listener for form button click
  - *call `generateGreeting()` function*
  - *pass `name` as argument to function, `generateGreeting(name)`*

```
// LISTEN: for user click on `greeting` button
greetingBtn.addEventListener('click', function() {
  // get name value from input field
    let name = document.getElementById('name').value;
    // get greeting message - pass input name...
    let greetingMessage = generateGreeting(name);

  // reset input field
    document.getElementById('name').value = '';
    // reset focus on input field
  document.getElementById('name').focus();
    // output greeting message to user
    document.getElementById('greeting').innerHTML = 'random greeting: ' + greetin
}, false);
```

HTML & JS - Random Greeting Generator

demo - update JS logic...

- Random Greeting Generator - Better 2

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

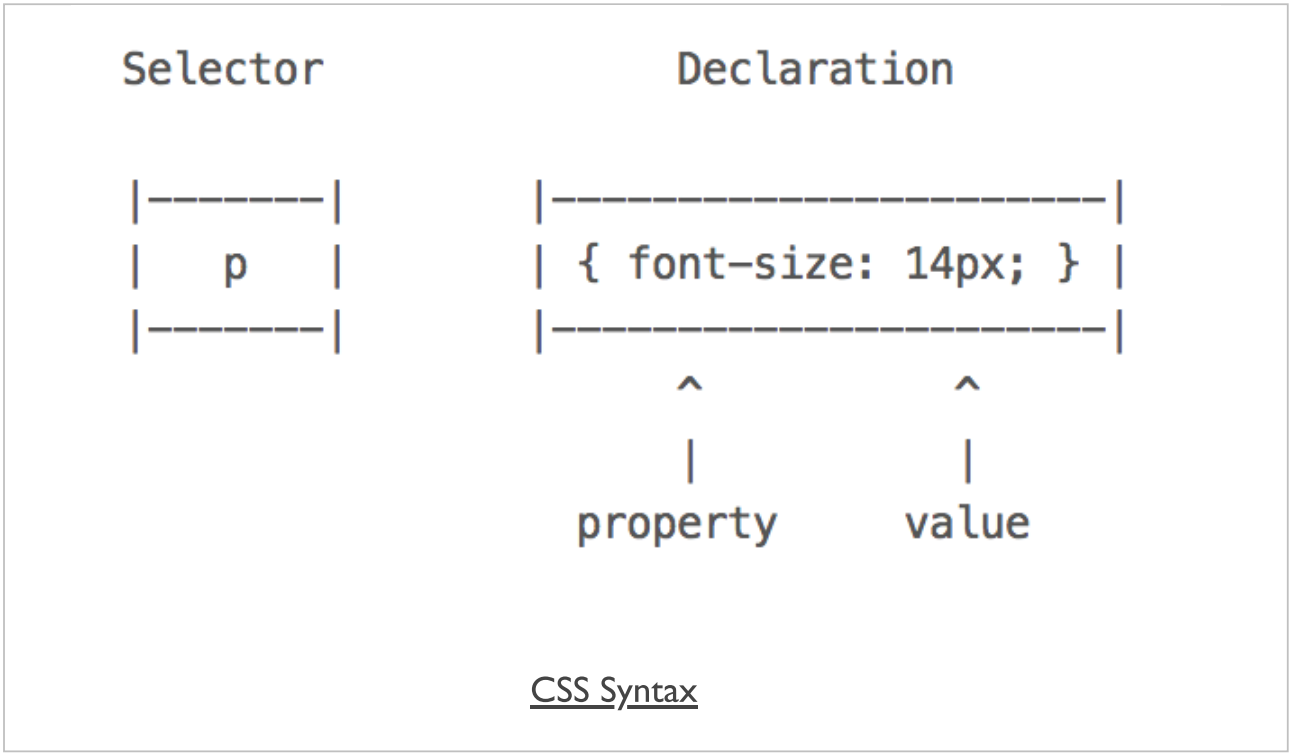
CSS Basics - comments

- 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

Image - CSS Syntax



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 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

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 I

- **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>`

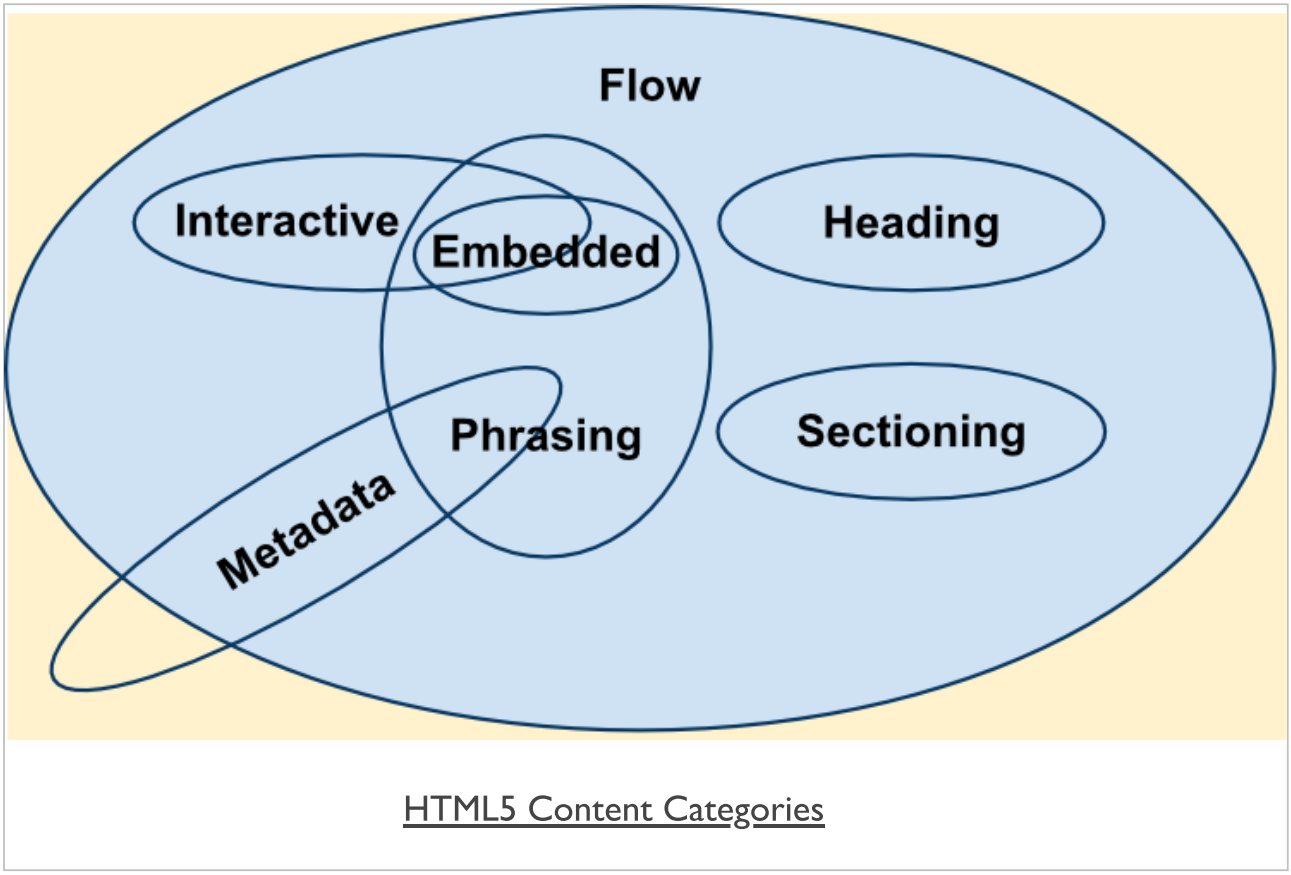
Source - MDN Content Categories

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