# Info Dump

Fast-paced intro to HTML & CSS Syntax

- A smidge of JS
- If new to HTML/CSS
    - VERY FAST and Shallow
        - Follow Readings+Resources to get more
    - Important highlights
- If experienced w/HTML+CSS
    - Pay attention
        - Details for this course are emphasized

# The Trinity of the Web

- **HTML** - The *structured content* of the page
    - WITHOUT regard to appearance
- **CSS** - The appearance of the content
    - Defined by structure
- **JS** - Interactions with the content
    - Other than navigation

Violating these roles can "work"

- But result will have limitations and problems

# What is HTML

- H - Hyper
- T - Text
- M - Markup
- L - Language

In other works, text that can link to other text, with "markup" in it to apply non-textual details.

```
This has a <a href="other-file.html">link</a> to another file
```

This has a **link** to another file

# Where does the HTML come from?

Web server response can contain HTML from:

- Static HTML - a file of HTML content
- Dynamic HTML - generated as program output

Not all web responses are HTML

- But web pages are all HTML
- Client (browser) doesn't know if dynamic or static

# HTML will refer to non-HTML assets

CSS, JS, Images

- All may impact display of Page
- But aren't HTML

References may be

- In the HTML document
- Or in some HTML elements
- Not preferred in general
- Not permitted for INFO6250

References may have URL for an asset

- Browser will make separate web request

# Browser Rendering an HTML Page

Rendering = Creating visual representation

**http://examplecat.com/**

- Figure out size and visual properties
    - Of every element "box"
- Download CSS/images/etc files
    - As references encountered
- Applying those files
    - Updating the sizing and visuals as needed
- Downloading/running JS as encountered
    - Modify the output-to-render as needed

# Semantic HTML

HTML is the structured content of the page

Think an organized list of everything in the page

- Like an outline, but with the text

You can try to use HTML for looks

- But that will fail
- Devices (mobile, desktop, versions) work diff
- Browsers show things differently
    - How does a paragraph, button, list look?

# What does "Semantic" mean?

"related to meaning"

Several words

- a paragraph?
- a heading?
- an item in a list?

It might be part of a navigation, or a section, or a link.

But these aren't APPEARANCE related.

Don't say where they appear or what they look like.

# Semantic Elements make Web Flexible

- Different browsers/devices can render content appropriately for their size
- Search engines can tell if a page is ABOUT cats vs having the word "cats"
- Special programs can interact with a page, filling in forms and performing actions

Only if the content uses **elements semantically**

- *Very* common mistake among developers

# HTML Tags

- The start/end indicators: **tags**
- Indicators + content: **element**

"tags"/"elements" are often used interchangeably

- Technically different

```
<a href="cats.html">More Cats</a>
```

A tag is a term in **angle brackets** `< >`

Tags should be **lowercase** text

# Opening and Closing Tags

A tag can be an "**opening**" or "**closing**" tag

- Closing tags begin with a slash `/` inside angles
  - `<p>This is a paragraph</p>`

An element can be "self-closing" (no content)

- `<img src="cat-with-glasses.png"/>`
- Some elements require content (open/close)
- Some elements don't (self-closing/empty/void)

# Weird Exceptions

A few elements feel like exceptions

- `<script></script>` MUST have open/close tags
    - Even when no contents
- Empty/void elements don't require a closing
    - But in HTML5 CAN optionally self-close
    - `<input>`
    - `<meta>`
    - `<img>`
    - `<br>` (Basically never use `<br>`!)

# The <br> element

- Used to create a visual line break
    - But that's not semantic!
        - Except for poetry
- Should almost never be used!
    - Except for poetry
- Does not require a close
- Has no content
- `<br>` or `<br/>`
    - **But you shouldn't be using it**
    - Spacing is not the job of HTML!

# Attributes

A tag can have **attributes**

- After tag name, before angle bracket
- `name="value"`
    - `<img src="cat.png" alt="Jorts the Cat"/>`
- Name without quotes
- Value with quotes
- (tradition) No space around the `=`
- (tradition) Double quotes (`"`) around the value
- This traditional syntax **required** for this course
    - Because Programming is Communication

# Empty Attributes

Some attributes don't have values

- Simply exist or do not exist
- Indicate boolean states
- Ex: `disabled`, `readonly`, `selected`

```
<input type="text" disabled/>
```

## Do not give these attributes values

`<input disabled="false"/>` - Still Disabled!

Just include the attribute or not

- Because the values are strings, not booleans

# References

Elements can refer to other files in different ways

This is annoying, but you just have to learn them

- `<img src="cat-wearing-hat.png"/>`
- `<a href="other-file.html">Link</a>`
- `<link href="file.css"/>`
- `<script src="file.js"></script>`

Always using URLs

- These are all **relative path** URLs
- Some elements use `src`, others use `href`

# HTML element ids

The `id` attribute identifies one exact element

- Value is a label with no technical meaning
- Unique per-page
    - Ex: Only one id "root" per page
- Only one `id` per element
    - Ex: Element w/id "root" has no other id
- Commonly used in direct HTML
- Commonly AVOIDED in dynamic HTML
    - Sometimes it is unavoidable

```
<div id="root">This is the root element</div>
```

# HTML element Classes

Elements can be identified by "**class**"

- No relation to programming concept (**None**)
    - This is "class" like "category"
- Many elements can have the same class
- An element can have many classes
- Multiple classes separated by spaces in value
- Order in the attribute value doesn't matter

```
<div class="selected example">A div with classes</div>
<div class="example">Another div on the same page</div>
```

- For INFO6250: kebab-case (or BEM) (**Required**)

# Capitalization Styles Matter!

- `squishedlowercase` (**most HTML attributes**)
    - ALL lowercase; words squished together
- `kebab-case` (**CSS properties**)
    - ALL lowercase; words hyphenated (`-`)
- `MixedCase` (**JS Components, JS Classes**)
    - Words squished together; each capitalized
- `camelCase` (**JS variables**)
    - Words squished together; each capitalized
    - First letter NOT capitalized
- `UPPER_SNAKE_CASE`/`CONSTANT_CASE` (**JS Constants**)
    - ALL uppercase; underscored (`_`) words

# What words to use for HTML classes

- HTML classes are used for CSS and JS
  - Sometimes call "CSS classes" for this reason
- Different conventions exist
  - We will use `semantic` and `kebab-case`
  - BEM style is fine if you know it
- Like with HTML semantics
  - **Semantic classes** name what they identify
  - NOT for the intended effect ("utility classes")
    - Semantic (Good): `review`, `selected`, `menu`
    - Utility (Not for us): `bold`, `red`, `left`

**INFO6250 requires semantic kebab-case class names**

# What is CSS

- (C)ascading (S)tyle (S)heets

A set of rules for appearance

- That apply in "cascading" layers
- Based on STRUCTURE
    - Elements
    - Classes
    - Structural Relationships (parent/child/etc)
    - Attributes
    - States (checked/hovered/etc)

# Rules and Selectors

A "CSS Rule" has

- **Selector(s)**
    - Deciding what elements are impacted
- A block of **declarations**
    - Setting the value of **properties**

Each declaration ends in a semicolon

```
p {
  font-family: sans-serif;
  text-align: center;
  font-size: 1.2rem;
  color: #BADA55;
}
```

# The difference sources of CSS

CSS applied to an element can come from:

- A separate file entirely (**recommended**)
- A `<style>` element within the HTML
- A `style` attribute of an HTML element ("inline")

The not recommended options "work"

- Are harder to find/maintain

Assignments will penalize you for "inline CSS"

- Make sure you know what that is!
- **Never use a `style` attribute in INFO6250**

# Setting CSS Properties

- Determine different visual appearances
- Some properties modify that element only
    - Example: `width`
    - Descendants can be IMPACTED
    - But don't have their property changed
- Some properties impact all descendants
    - Example: `color`

Generally:

- Positioning and sizing don't **inherit**
- Typography and color do **inherit**

# Selectors

- HTML ids `#root { color: aqua; }`
- Element type `p { color: #C0FFEE; }`
- Classes `.wrong { color: red; }`
- Combinations `p.wrong { color: red; }`
- Descendants `.wrong p { color: red; }`
- Children `.wrong > p { color: red; }`

Any mix of the above, plus less common selector types

Applying rules based on descendants is more involved

**Selectors ultimately match elements**

# Which number(s) is/are red for each example?

```
<div class="css-example">
    1
    <p>2</p>
    <p class="example simple">3</p>
    <p class="example">4</p>
</div>
```

- `p { color: red; }`
- `.css-example { color: red; }`
- `.example { color: red; }`
- `p.example { color: red; }`
- `.css-example .simple { color: red; }`
- `.css-example.simple { color: red; }`

# Which number(s) was/were red?

- `p { color: red; }` **2, 3, 4**
  - Each `<p>` was matched
- `.css-example { color: red; }` **1, 2, 3, 4**
  - The `<div>` was matched, color was inherited
- `.example { color: red; }` **3, 4**
  - Each `class` with "example" was matched
- `p.example { color: red; }` **3, 4**
  - Each `<p>` that had `class` with "example"

# Which number(s) was/were red?

- `.css-example .simple { color: red; }` **3**
  - Each `class` "simple" that was a descendant of an element of `class` of "css-example"
  - `.css-example` element is NOT made red
  - 4 doesn't inherit from a sibling
- `.css-example.simple { color: red; }` **None**
  - No element has both `class` "css-example" AND `class` "simple"
- `.css-example .simple` vs `.css-example.simple`!

# Specificity

What if many rules can apply to an element?

- The "Cascade" of CSS decides which rules apply
- Rules have **Specificity**
- More Specific rules override less specific rules
- `style` attribute considered most specific
    - One reason we don't use

Better Summary:

- **https://2019.wattenberger.com/blog/css-cascade**

# Cascade and Specificity

1. Declarations marked `!important` win (**don't do**)
2. Inline CSS on the element wins (**don't do**)
3. The more specific selector wins
    - id(`#`) is most specific
    - class(`.`) less so
    - tag type is least specific
    - totals combine, so `.some.class` is twice as specific as `.class`
4. If all equal, "latest" rule overrides older rule
    - "latest" means later on the file/page

# What decides the color of Cat?

```
<div class="example">
    <p id="jorts" class="cat">Cat</p>
</div>
```

```
.cat {
    color: black;
}

#jorts {
    color: orange;
}

p {
    color: red;
}

p {
    color: green;
}
.example {
    color: blue;
}
```

# What decided the color of Cat?

```
<div class="example">
    <p id="jorts" class="cat">Cat</p>
</div>
```

Cat is **orange**

- `.example` sets the inherited color
    - Overridden by color on the actual element
- `p` selector is least specific (trying for **red**)
    - Second `p` overrides first (trying **green**)
    - Both overridden by more specific selectors
- `.cat` selector is more specific than `p` (trying **black**)
- The **id** selector was the most specific
    - Cat is **orange**

# Exception to "don't use"

Okay to use inline CSS (`style` attribute) on element if

- You're making changes via JS **AND**
- Those changes have unknown values in advance

`style` attribute (inline CSS) Okay:

- Changing size by dragging a mouse

`style` attribute (inline CSS) Not Okay:

- Setting an element to hidden/not hidden

# You will see MANY examples of Inline CSS Online

- Lots of tutorials and examples use it
- It "works" (**Working is Not Enough**)
- More common in component-based apps (React)
- HTML Inline CSS doesn't scale well with changes
    - Hard to read/change in larger code base
        - Doesn't have to be huge code base!
        - Just a few hundred lines is enough
    - Difficult to override
- **Do not use inline CSS (`style` attr) in this course**
    - You will miss important skills

# What is Javascript (JS)

Both JS, but notable differences!

- JS on the browser
- JS on the server

Work to understand differences

- Mostly the same syntax
- Differences in what they do
- And when they do it
- And on which computer they run

# JS in the browser

JS in the browser

- Sent from server
- Runs in the browser
- On THEIR machine (not on the server!)
- Knows only the data in this JS and in the page
- Can change the HTML on the rendered page
- Can add in reference to more CSS or JS
- Completely visible to the user

JS is the only (real) option to run in the browser

# JS on the server

Code running on the server can be in any language

- JS not special here like it is on the browser
- May generate unrendered HTML as text

For us JS is just convenient for the same language

- No access to the rendered page
- No awareness of what user is "doing"
- Server can only respond to requests

JS on server vs browser are completely disconnected

# Summary

- The different roles of HTML, CSS, JS
- What is semantic HTML
- Dos and Do Nots for element class names
- Different kinds of CSS selectors
- CSS rules of Specificity
- Diff between server-side JS and client-side JS

# Summary - Requirements for this Course

In and out of this course:

- HTML used semantically
- HTML boolean attributes have no values

In this course (and I recommend outside):

- HTML attributes with no spaces around `=`
- HTML attributes with double quotes around value
- CSS class names are semantic
    - and kebab-case or BEM

```
<input name="street" class="address" disabled />
```