HTML, CSS, and DOM Manipulation

CIS 1962 (Winter 2026)
Feburary 5th, 2026

Penn Engineering

Lesson Plan

Web Development

5	Intro to Web Development

HTML

9	What is HTML?
14	HTML Tags and Attributes

CSS

24	What is CSS?
30	CSS Selectors
36	CSS Properties

DOM Manipulation

51	DOM and Selectors
59	Changing DOM Content
67	Event Listeners

Weekly Logistics

Homework 2: Project Scaffolding DUE TONIGHT

Homework 3: Echo Chatbot RELEASED

- Due on February 12th @ 11:59 PM through GitHub Classroom!
- This is an open-ended assignment in terms of web page design- you will be writing most of the HTML/CSS yourself to match classic chatbots!
- This is also a prequel to HW4- you'll be actually building this app out into a full chatbot in that one!

Before we start: VSCode Setup

In this lecture, we will be learning to write HTML, CSS, and JavaScript code to create webpages.

You can follow along using Ed Lessons, or setup your IDE to serve up HTML.

We recommend you use the **Live Server** VSCode Extension to serve up your webpage.

You can also use **npm serve** package to easily serve your static site.

- Install the package with `npm i serve`
- Run the command `npx serve .` to display your HTML (make sure your html is called `index.html`!)

Web Development

What is web development,
and how is code used to build a website?

Web Development

In the modern age, you cannot escape the web. From social media websites, to school websites or the menus of local businesses, to E-commerce and tools like Google Drive and ChatGPT.

The foundation of all these websites, no matter what new technology arises, remains **HTML, CSS, and JavaScript**.

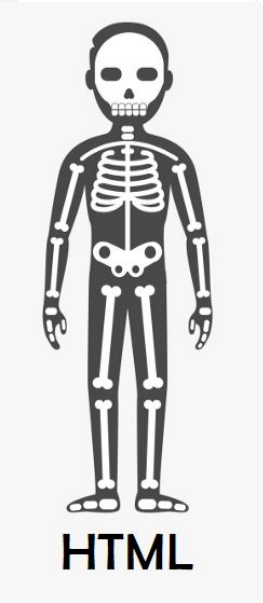
Web Page Rendering

Browsers process **HTML** to structure the page by building a DOM (Document-Object Model).

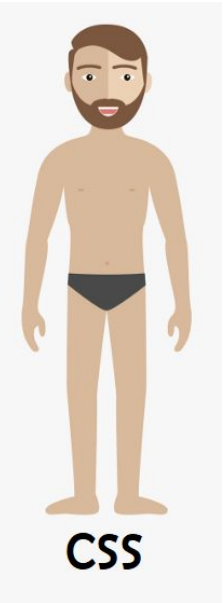
They then apply **CSS** for styling, such as colors, layout, fonts, hovering, and mobile adjustments, among others.

The **JavaScript** engine then adds the behavior and interactivity of the website.

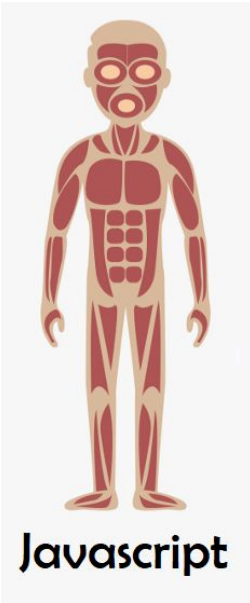
A Web Page is like a human body…



Bones



Skin & Clothes



Muscular/Nervous System

HTML

How do we define the structure of a webpage?

HyperText Markup Language (HTML)

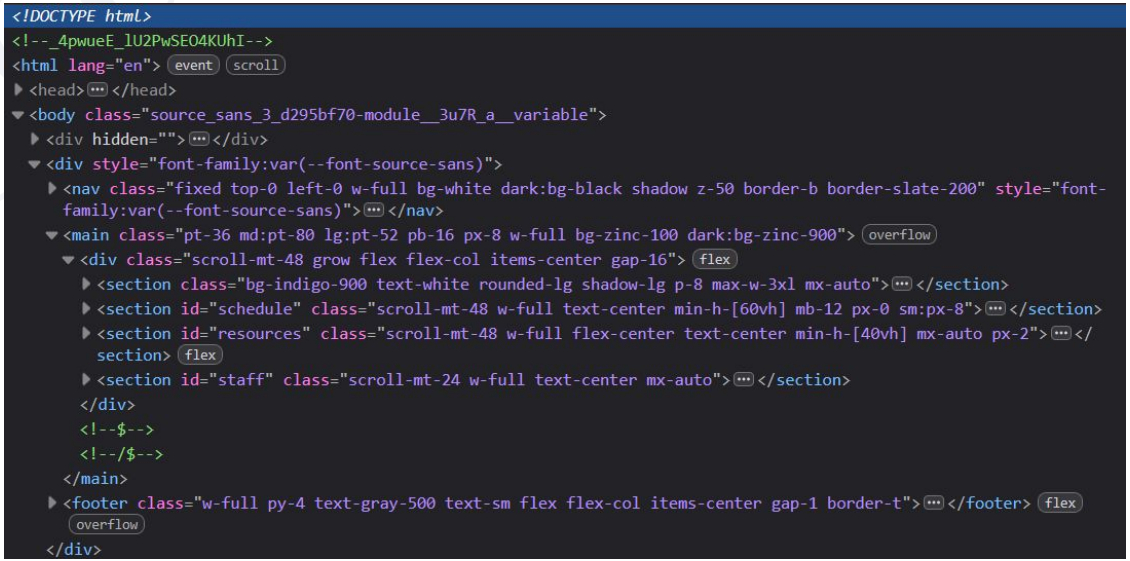
HyperText Markup Language, abbreviated **HTML**, defines the structure of web pages through tags <> and nesting to define where elements exist on a webpage.

A web page is built up of elements defined by tags like a tree, with parents having child elements, leading up to a root.

Let's study some HTML!

Open the class website (https://www.cis.upenn.edu/~cis1962/)

Inspect the HTML through Browser Dev Tools:



(the above view comes from Mozilla Firefox's dev tools, your view may differ slightly)

Basic HTML Skeleton

Every HTML has the basic skeleton:

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <!-- metadata goes here -->
     </head>
     <body>
          <!-- content goes here -->
     </body>
</html>
```

<html>: the root of the HTML document, containing all other HTML elements inside it. Also used to declare the language of the page.

<head>: Used to define metadata about a page, including titles, css, scripts, and page details.

<body>: Used to define the visible content of the page.

Metadata

Metadata is the information describing a page. It may include:

- CSS Files
- Scripts
- Search Engine Optimization (SEO) information
- Accessibility tools
- External links to resources, such as fonts

HTML Tags

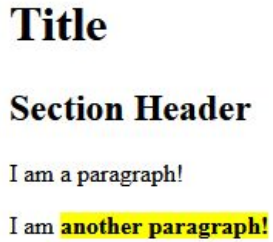
HTML tags are the building blocks of a web page, guiding a browser on where and how to orient and display text, images, links, and other media.

- **Grouping:** `<div>, <span>, <section>`
- **Text:** `<p>, <h1>` thru `<h6>`
- **Links:** `<a href="...">`
- **Images:** `<img src="..." alt="..." >`
- **Lists:** `<ul>, <li>, <ol>`
- **Navigation:** `<nav>, <header>, <footer>`
- **Input:** `<button>, <input>, <form>, <label>`

HTML Tags: Grouping & Text

Tag	Description	Example
`<div>`	A generic divider for grouping content and layout	
`<span>`	Inline container, often used to apply different styles inline	
`<h1>, <h2>... <h6>`	Section headings to form the outline of a document	
`<section>`	Logical sections of a document	
`<p>`	Paragraph	

```
<div class="homepage">
   <h1> Title </h1>
   <h2> Section Header </h2>
   <section class="section1">
      <p>I am a paragraph!</p>
      <p>I am <span class="highlight"> another
paragraph! </span></p>
   </section>
</div>
```

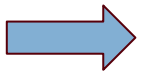


HTML Tags: Links & Images

Tag	Description	Example
<a>	Anchor, defines hyperlinks with href attribute	 Visit me! <img src="https://placecats.com/300/200" alt="A cute cat">
<img>	Show images, requires an src attribute with a link to the image. Can also specify alt text for screen readers with the alt attribute.	

*Both <a> and <img> are inline elements, thus they won't stack vertically when placed together

```
<a href="https://example.com">
   Visit me!
</a>
<img
src="https://placecats.com/300/200"
alt="A cute cat">
```



HTML Tags: Lists and Navigation

Tag	Description	Example

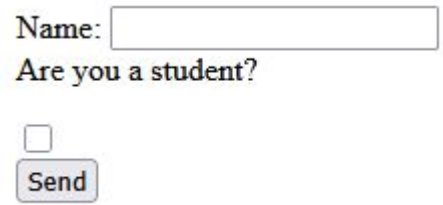
`<ul>,<ol>,<li>`	Unordered/ordered lists and list items	`<header>`
		`  <h1>My Sample Page</h1>`
		`  <nav>`
		`  <ul>`
		`  <li>`
		`  <a href="#about">About </a>`
		`  </li>`
		`  <li>`
		`  <a href="#contact">Contact</a>`
		`  </li>`
		`  </ul>`
		`  </nav>`
		`</header>`

`<nav>`	Navigation section, for site menus	
`<header>, <footer>`	The top and bottom sections of a site	

HTML Tags: User Input

Tag	Description	Example
`<button>`	Interactive element that can be activated with various interactions to perform an action.	
`<input>`	An element that contains a wide variety of user input widgets to acquire user input data.	
`<form>`	A document section for submitting information.	
`<label>`	Inline captions for items in interfaces	

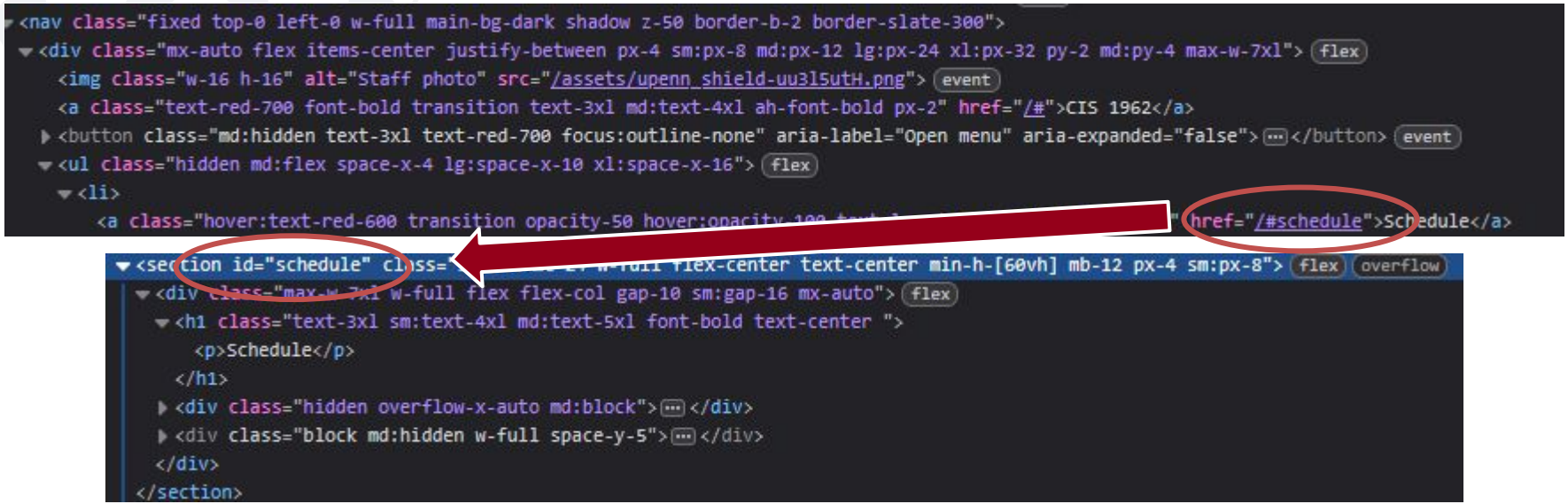
```
<form>
   <div>
      <label for="name">Name:</label>
      <input type="text" name="name"/>
   </div>
   <div>
      <label for="checkbox">Are you a
student?</p>
      <input type="checkbox"
name="checkbox"/>
   </div>
   <button type="submit">Send</button>
</form>
```



Attributes: href and id

You can add **attributes** within the angle brackets of tags, such as the **id** attribute.

An <a> tag can specify an **href** attribute that hyperlinks to other pages. It may also hyperlink to the same page using "**#**" and pointing to the id of another tag.



Semantic & Non-Semantic Tags

Non-semantic tags, such as and say nothing about their content.

Semantic HTML tags, all other tags, describe the purpose of their content, and help improve SEO and accessibility.

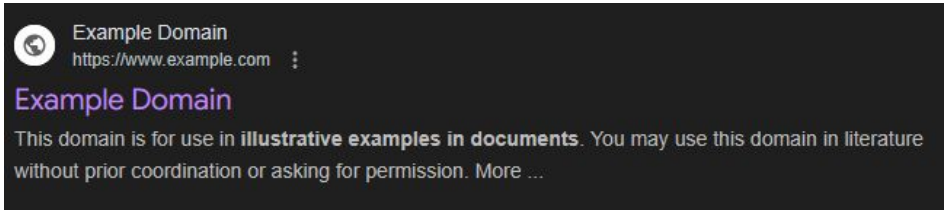
```
<div class="big"> Neo the Cat </div>
<div> Neo is a cute cat! </div>
<div>
     <img
src="https://placecats.com/neo/300/200">
</div>
```

```
<h1>Neo the Cat</h1>
<p>Neo is a cute cat!</p>
<img src="https://placecats.com/neo/300/200">
```

Why use proper semantic tags?

It's important to use proper tags for reusability, readability, and accessibility!

Search engines and screen readers heavily rely on good use of HTML tags on web pages.



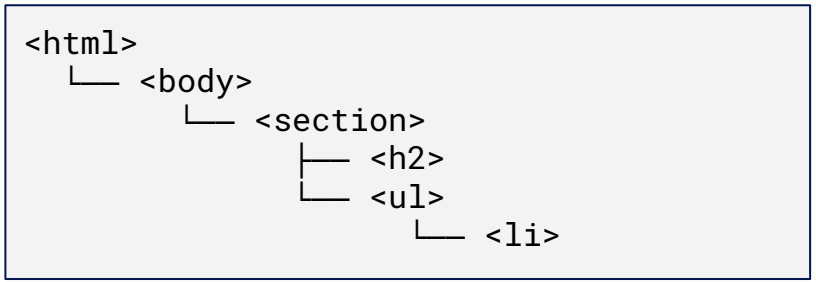
example.com

Nesting of HTML Elements

If an HTML tag contains other tags in it, the inside elements are children of the parent element.

You can make use of this to group sections together and apply attributes/css selectively.

```
<html>
  <body>
    <section>
      <h2>My Hobbies</h2>
      <ul>
        <li>Art</li>
      </ul>
    </section>
  </body>
</html>
```



Live Coding Activity: Make this Profile!

Replicate the image to the right using only HTML!

Pick a cute cat photo here and use it as the <img> src: https://placecats.com/, or use your own image!

Attach "mailto:" in front of an email address in an <a> href to turn it into an email link!

CSS

How do we apply colors, layout, text features, and other style features to a webpage?

What is CSS?

You can apply styles to HTML using various attributes, such as "style".

```
<p style="color:red">I'm red!</p>
<p style="font-size:20px">I'm big!</p>
```

This method, however, is very messy and hard to maintain.

CSS (Cascading Style Sheets) was invented to separate the development of structure and style of web pages.

Creating CSS Styles

Inline

Generally not recommended unless you have specific style overrides you want to apply.

```
<p style="color:red">
     I'm red!
</p>
```

Internal Stylesheet

Using the <style> tag in the <head>, an internal stylesheet can be maintained.

```
<head>
  <style>
    p { color: green; }
  </style>
</head>
```

External Stylesheet

This method is generally the best practice, and even allows multiple stylesheets to be applied to HTML documents.

```
        (in main.html)
<head>
  <link rel="stylesheet"
href="styles.css">
</head>
        (in style.css)
p {
  color: purple;
}
```

Anatomy of a CSS Rule

CSS works by applying "rules" to parts of a document.

```
selector {
    property: value;
}
```

The **Selector** determines **what part** will be styled.

The **Property** determines **what aspect** will change.

The **Value** determines **how** the property changes.

Example CSS Rules

```css
h1 {
  color: blue;
  font-size: 32px;
}
```

Change the color of all text in <h1> tags to blue and their font size to 32px

```css
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

Change the background color of elements with the class "highlight" to yellow, and make their text bold.

```css
ol li {
  font-style: italic;
}
```

Style any <li> elements inside an <ol> element to be italicized.

The "class" attribute

You will most often be using the "class" attribute within HTML tags to apply CSS rules, while writing the definitions of those classes within a stylesheet.

```
<p class="highlight">Highlight me!</p>
```

```
.highlight {
  background-color: yellow;
  font-weight: bold;
}
```

Highlight me!

CSS Selectors

Element: target all elements of a given tag.

```
div {
     display: flex
}
```

.class: targets all elements with a certain class

```
.important {
     font-weight: bold;
}
```

#id: targets an element with a unique ID

```
#schedule {
     text-align: center
}
```

Descendant: target specific child elements inside other elements

```
section ul li {
     list-style-type: none
}
```

CSS Selector Example

```css
h1 {
  color: darkgreen;
}

#spicy-meatball {
  font-style: italic;
}

.big-text {
  font-size: 32px;
}

div p {
    color: crimson
}
```

```html
<h1>Ooooh mama mia</h1>
<p id="spicy-meatball">That's one <span
class="big-text">spicy</span> meatball!!!</p>
<p>This is a list of my secret ingredients:</p>
<div>
     <p>Super Fresh Basil</p>
     <p>3 Entire Carolina Reapers</p>
</div>
```

Cascading Styles

Cascade: If multiple styles apply to an element, what style wins?

Three main principles, ordered by what applies with priority (left = most priority):

- **Source**: Inline Style **>** Internal stylesheet **>** External stylesheet
- **Specificity**: Inline Style **>** #id style **>** .class style **>** element style
- **Importance:** If a property is marked with !important, it will override almost everything else.

Cascading Styles: Example

What color will the text of the element below be?

```
<p id="greeting" class="highlight">Wahoo!</p>
```

```
p {
  color: blue;
}
.highlight {
  color: green !important;
}
#greeting {
  color: orange;
}
```

Pseudo-Classes

Pseudo-classes are keywords applied to selectors to affect specific states of elements. They are denoted by a colon (:) and a name after a rule.

For instance:

- `:hover` lets you define what happens when a user hovers their mouse over an element
- `:visited` targets links that are already visited
- `:first-child` targets the first element that is the first child of the selected element.

Pseudo-Classes: Hover Example

```css
a {
    color: #1976d2;
    font-size: 32px;
    text-decoration: none;
    font-weight: bold;
}

a:hover {
    color: #f39c12;
    text-decoration: underline;
    cursor: pointer;
}
```

Hover over me!

Hover over me!

Important CSS Properties

- **Colors:** `color`, `background-color`
- **Borders:** `border`, `border-radius`
- **Text:** `font-size`, `font-weight`, `font-family`, `text-align`
- **Spacing:** `margin`, `padding`
- **Layout:** `display`, `width`, `height`, `justify-content`, `align-items`, `flex-direction`

CSS Properties: Color

Colors: `color, background-color, border-color`

Values can be pre-defined color values (like navy, floralwhite, or firebrick), 3 RGB values, or #hex codes.

```
p {
    color: #dbd8d8;
    background-color: navy;
    border-style: solid;
    border-color: rgb(190, 31, 31);
}
```

CSS Properties: Borders

Borders: `border, border-radius`

Border uses the the value format: `<width> <style> <color>`

Border radius defines how rounded the corners are.

```
img {
     border: 4px solid aqua;
     border-radius: 10px;
}
```

CSS Properties: Text

Text: font-size, font-weight, font-family, text-align

```css
h1 {
  font-size: 32px;
  font-weight: bold;
  font-family: 'Courier New', monospace;
  text-align: center;
}

p {
  font-weight: 300; /* thin */
  font-family: Arial, Helvetica, sans-serif;
  text-align: right;
}
```

IMPORTANT ANNOUNCEMENT

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Duis non fermentum lacus. Nunc quis lobortis justo, sed interdum mi. Integer faucibus nibh pretium mauris dictum consequat. Morbi imperdiet faucibus imperdiet. Suspendisse turpis nisi, suscipit non turpis in, mollis tincidunt quam. Pellentesque a est est. Nam dignissim feugiat est, sed maximus erat.

CSS Properties: Spacing

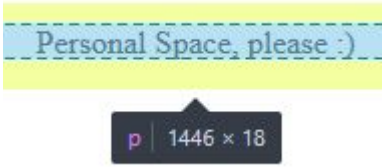
Spacing: `margin, padding`

You can manipulate the margin/padding in all 4 directions.

```
div {
     padding: 15px;
}

p {
     text-align: center;
     margin-top: 10px;
     margin-right: 5px;
     margin-bottom: 15px;
     margin-left: 5px;
}
```

margin



padding

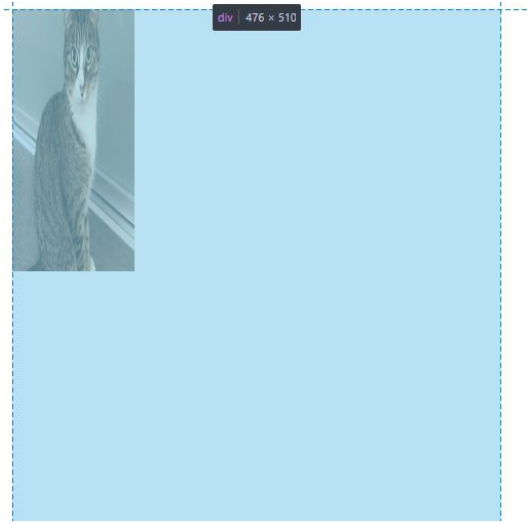
CSS Properties: Width/Height Layout

Layout: `width, height`

For height and width values, you can specify % values to scale based on a % of a parent, or the unit vh, to scale based on the visible area of the webpage.

```
img {
  width: 25%;
  height: 50%;
}

div {
  width: 25%;
  height: 100vh;
}
```



CSS Properties: Display & Flexbox

Layout: `display, justify-content, align-items, flex-direction`

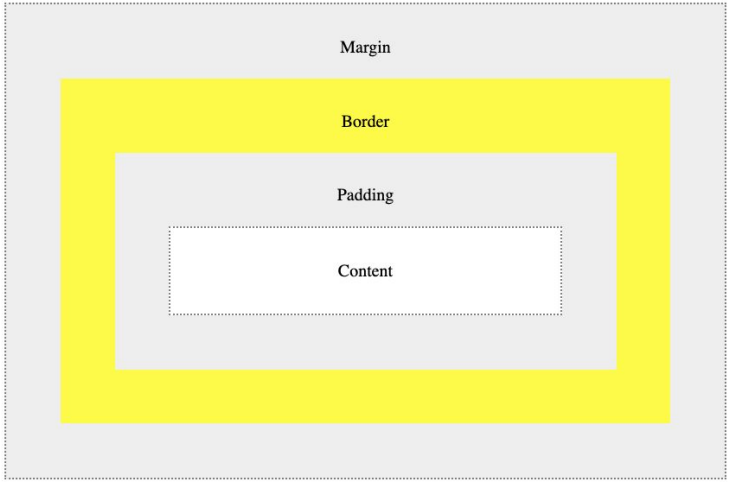
Display allows you to, for instance, turn an item into an inline item. Among these display options is `display: flex`, allowing you to use the versatile **Flexbox**.

```
.col-spread {
  display: flex;
  justify-content: space-between;
  color: white;
  padding: 20px
}
```

The Box Model

The box model describes how every element in CSS is a rectangular box consisting of:

- **Content**: The actual content (text, images, etc.)
- **Padding:** Space between content and border
- **Border:** The border around the padding
- **Margin:** Space outside the border



Flexbox

Flexboxes are a popular way to layout items on a webpage, in a one-dimensional form (horizontally or vertically) that is responsive (responds to different screen sizes)

The element containing `display: flex` will become a **flex container**, with direct children becoming **flex items**.

```
.col-spread {
  display: flex;
  justify-content: space-between;
  color: white;
  padding: 20px
}
```

Flexbox: justify, align, and gap

`justify-content` aligns items along the **main axis**.

`align-items` aligns items along the **cross axis** (perpendicular to main axis).

- `center`: items centered
- `space-between:` equal space between items
- `space-around:` equal space around items
- `space-evenly:` equal space between and around items

`gap` adds space between flex items.

Flexbox: flex-direction

`flex-direction` defines the main axis, the directions items will flow.

- `row:` left to right (default)
- `row-reverse:` right to left
- `column:` top to bottom
- `column-reverse:` bottom to top

When you flip the direction from row to column, remember that justify and align will flip directions as well!

Honorable Mention: Flexbox Froggy

https://flexboxfroggy.com/

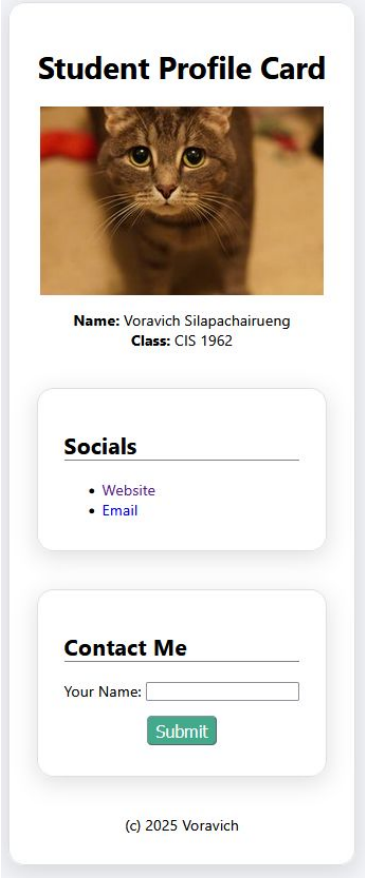
This site will give you great practice with

Flexboxes and CSS layout!

Live Coding Activity: Style your Profile

Let's add some style to the profile card we made earlier.

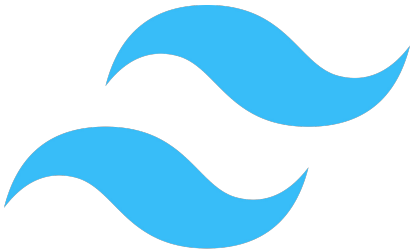
- Try **centering** most of the items (text-align: center , margin: auto)
- Look into the `box-shadow` property for the entire card and its sections (documentation)
- Consider affordances: what happens when you **hover** over links or buttons?
- Try putting everything in a flexbox!



CSS Frameworks

Several pre-written libraries already exist to help developers quickly style pages and build websites.

These are CSS frameworks such as **Bootstrap, Bulma, Tailwind, and Foundation**. Most include pre-written classes that can be applied to unify the look of an app quickly.



5-Minute Break!

DOM Manipulation

How do we use JavaScript to affect elements on the page and make web pages dynamic?

Document Object Model

As you may recall, the browser represents HTML as a tree of elements: The **Document Object Model (DOM)**.

The hierarchy of tags/elements are represented as nodes and their children.

In order to change the DOM, you must use DOM API methods, called from a `document` object that refers to the DOM.

DOM Manipulation Example

```
<p id="greeting">Hello Everyone!</p>
```

Hello Everyone!

```
const greeting = document.getElementById('greeting');
greeting.textContent = "How are you all doing?"
```

How are you all doing?

JS Scripts & HTML

JavaScript can be attached internally to HTML:

```
<body>
      <script>
            const greeting = document.getElementById('greeting');
            greeting.textContent = "How are you all doing?"
      </script>
</body>
```

You can also attach external scripts:

```
<body>
      <script src="test.js"></script>
</body>
```

Selecting Element By ID

The DOM API provides ways to select elements for manipulation. One popular one is **selection by ID**.

```
<p id="greeting">Hello Everyone!</p>
```

Hello Everyone!

```
const greeting = document.getElementById('greeting');
```

Selecting Elements By Tag/Class

You can also select by tags or classes:

```
<p class="highlight">Meow</p>
<img src="https://placecats.com/300/200"/>
<p class="highlight">Meow</p>
```

```
const highlighted = document.getElementsByClassName('highlight');
const image = document.getElementsByTagName('img');
```

Meow

Meow

Note that this method always returns a **collection** of items, not just one.

Query Selectors

The `querySelector()` method is a versatile selector that returns the **first** element that matches a certain CSS selector.

```
<div id="cats">
    <p class="highlight">Meow</p>
    <img src="https://placecats.com/300/200"/>
    <p class="highlight">Meow</p>
</div>
```

```
const catsP = document.querySelector('#cats p');
```

You can also use `querySelectorAll()` to get all instances that match the selector.

Become an (Internet) Time Wizard

With knowledge of DOM manipulation, you unlock secret powers with internet videos!

A <video> has a property called '`playbackRate`'. If your video player doesn't have playback rate button/option, you can set it yourself!

In Dev Tools, try:

```
document.querySelector('video').playbackRate = 2
```

Properties: Text Content & InnerHTML

`element.textContent`: Controls the text inside an element, though it is not always visible (i.e. <img>, <input> don't show text)

`element.innerHTML`: Controls actual HTML contents between tags, allows insertion of new children in hierarchy

Text Content & InnerHTML Examples

```
<p id="demo">Hello <b>world!</b></p>
```


```
const el = document.getElementById('demo');
console.log(el.textContent);
// Output: "Hello world!"
```

```
const el = document.getElementById('demo');
console.log(el.innerHTML);
// Output: "Hello <b>world!</b>"
```

!!! The Dangers of innerHTML !!!

`innerHTML` introduces a possible security vulnerability into your app called **Cross-Site Scripting (XSS)**.

This is when some user input is allowed to run code, usually through some untrusted user input!

This can do things like steal session cookies, redirect to malicious websites, or perform unintended actions on a website.

```
const userInput = '<img src="x" onerror="alert(document.cookie)">';
container.innerHTML = userInput; // Attacker's code runs!
```

Properties: Changing Attributes

`element.[attributeName]`: Allows you to directly change attributes of tags, such as `src`, `href`, or `alt`.

```
<img src="https://placecats.com/300/200" alt="cute cat"/>
```

```
document.querySelector('img').src = 'https://placecats.com/300/300';
document.querySelector('img').alt = 'cute cat in 300 x 300 size';
```

Properties: Changing Style

`element.style`: Allows you to change the style of an element.

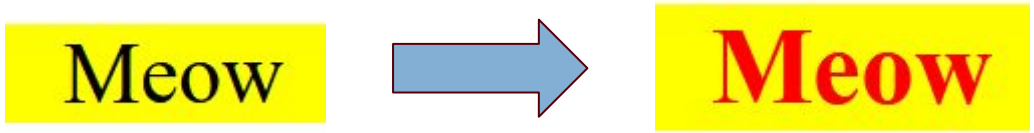
`element.classList`: Includes various methods to add CSS classes to the `class` attribute, including:

- `element.classList.add()`
- `element.classList.remove()`
- `element.classList.toggle()`: Remove if present, add if not present

Changing Style Example

```
<p class="highlight">Meow</p>
```

```
document.querySelector('p').style.color = 'red';
document.querySelector('p').style.fontSize = '18px';
document.querySelector('p').classList.toggle("bold-text")
```



Adding/Removing Elements

`document.createElement()` can be used to create an HTML element within JavaScript. However, it doesn't exist within the actual HTML until properly added.

`element.appendChild()` allows you to add a previously created element as the last child of a specified element.

`element.remove()` removes the child from HTML, regardless of where it is in the DOM.

Adding/Removing Elements: Example

```html
<div id="container"></div>
<button id="addBtn">Add Item</button>
<button id="removeBtn">Remove Item</button>
```

```javascript
const container = document.getElementById('container');
const addBtn = document.getElementById('addBtn');
const removeBtn = document.getElementById('removeBtn');

addBtn.addEventListener('click', function() {
  const newItem = document.createElement('div');
  newItem.textContent = 'I am a new item!';
  newItem.className = 'item';
  container.appendChild(newItem); // Add to DOM
});

removeBtn.addEventListener('click', function() {
  const lastItem = container.querySelector('.item:last-child');
  if (lastItem) {
    container.removeChild(lastItem); // Remove from DOM
  }
});
```

What are Events and Event Listeners?

Events are actions or occurrences that happen in the browser, triggered by a user or a system. These include:

- User clicking a button
- A form is submitted
- A specific key is pressed
- A mouse hovers over something

Browsers provide a Web API to handle events called **event listeners**.

Event-Driven Programming

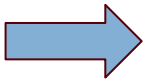
Web pages are no longer just static- they need to be able to update and react to user and system feedback.

Event-driven programming is a style of programming where the flow is determined by often asynchronous events, rather than linear execution.

Handling Events

We refer to the action of running code in response to something happening as **"handling"** an event.

We specify **handlers** that will wait for some action to occur, and then execute when that action occurs.



```
btn.addEventListener('click', function()
{
  btn.style.backgroundColor = 'red';
});
```

Handler executes

Event Listener: Syntax

```
element.addEventListener('eventType', handlerFunction);
```

element: HTML element that this listener waits on an event for

eventType: The event type, such as ‘click’ or ‘hover’

handlerFunction: A callback function that gets called when the event happens. Often allows use of the ‘event’ object as an argument

Event Listener: Syntax (internal)

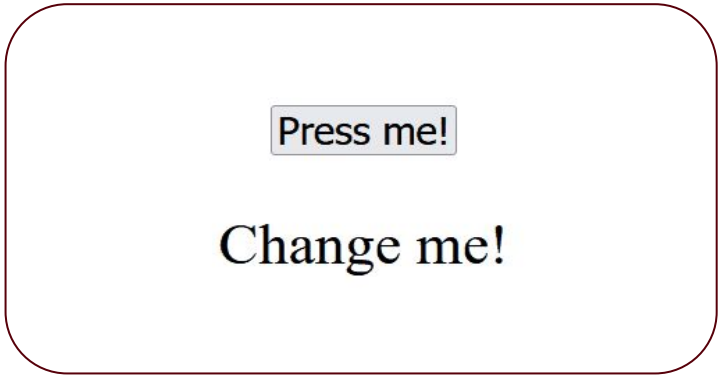
```
<button class="submit" onclick="submitForm()">Submit</button>
```

You can define event handlers internally within HTML.

This is good for simple events, but not good for any more complex contexts.

Event Types: Clicking

```
document.getElementById("demo-btn")
    .addEventListener('click', () => {
        document.getElementById("output")
            .textContent = "Meow"
    })
```



Event Types: Hover (Enter/Exit)

```
document.getElementById("demo-btn")
    .addEventListener('mouseenter', () => {
      document.getElementById("demo-btn")
        .classList.add("fancy-button")
    })
```

```
document.getElementById("demo-btn")
    .addEventListener('mouseleave', () => {
      document.getElementById("demo-btn")
        .classList.toggle("fancy-button")
    })
```

Event Types: Focus (In/Out)

```
document.getElementById("focus-input")
    .addEventListener('focusin', (e) => {
       document.getElementById("focus")
         .textContent = `Focusing on input!`
    })
```

```
document.getElementById("focus-input")
    .addEventListener('focusout', (e) => {
       document.getElementById("focus")
         .textContent = `Awaiting focus...`
    })
```

Event Types: Typing ('keydown')

```
document.getElementById("await-input")
    .addEventListener('keydown', (e) => {
        document.getElementById("await")
         .textContent = `You typed ${e.code}`
    })
```

The event object

The handler function can receive an event object as an argument. While this is optional, it contains important information about the event, such as:

- What triggered the event?
- What key was pressed?
- What element was clicked?
- When was it clicked?


```
document.getElementById("await-input")
    .addEventListener('keydown', (event) => {
        document.getElementById("await")
            .textContent = `You typed ${event.code}`
    })
```

Propogation/Bubbling

When some element triggers an event, the event "bubbles" up the DOM tree until the root. This means that multiple handlers in the same branch will trigger. You can stop this with `event.stopPropogation()`.

```
<div id="parent" style="padding:40px; background:lightblue;">
     Parent Div
     <button id="child">Click Me</button>
</div>
```

```
document.getElementById("parent").addEventListener("click", () => {
     console.log("Parent div clicked");
});

document.getElementById("child").addEventListener("click", () => {
     console.log("Button clicked");
});
```

event.preventDefault()

Some events have default actions that are undesirable. For instance, submitting a form (with the event 'submit') will usually refresh a page.

You can prevent this using event.preventDefault().


```
document
  .getElementById('loginForm')
  .addEventListener('submit', function(event) {
    event.preventDefault(); // Prevents page refresh
  });
```

Live Coding Activity: Make it Interactive!

We'll do some things to make your profile card interactive:

- Add a rudimentary dark mode toggle. This means a button that will apply a new style across the entire card!
- Add a "change name" button and input below your name, so that you can dynamically change your name.
- Prevent the submit form from refreshing the page!