HTML, CSS, and DOM Manipulation

CIS 1962 (Fall 2025)
September 29th, 2025

Lesson Plan

Weekly Logistics

Homework 1: Data Analysis DUE MIDNIGHT TONIGHT

Homework 2: ChatJS RELEASED

- Due on October 20th @ 11:59 PM through GitHub Classroom!
- You have 3 weeks for this assignment, pace yourself, there are clear checkpoints that you should hit during those weeks!
- 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 LLM chatbots!

Next Week: 2nd Half Lecture Content Survey!

Before we start: VSCode Setup

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

We recommend you use the **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`!)

You can also use the **Live Server** VSCode Extension to serve up your webpage.

PollEverywhere!

We will use PollEv to take attendance and do polls during lecture. Scan this QR Code or use the link to join for attendance! (Please add your name for identification)

PollEv.com/voravichs673

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.

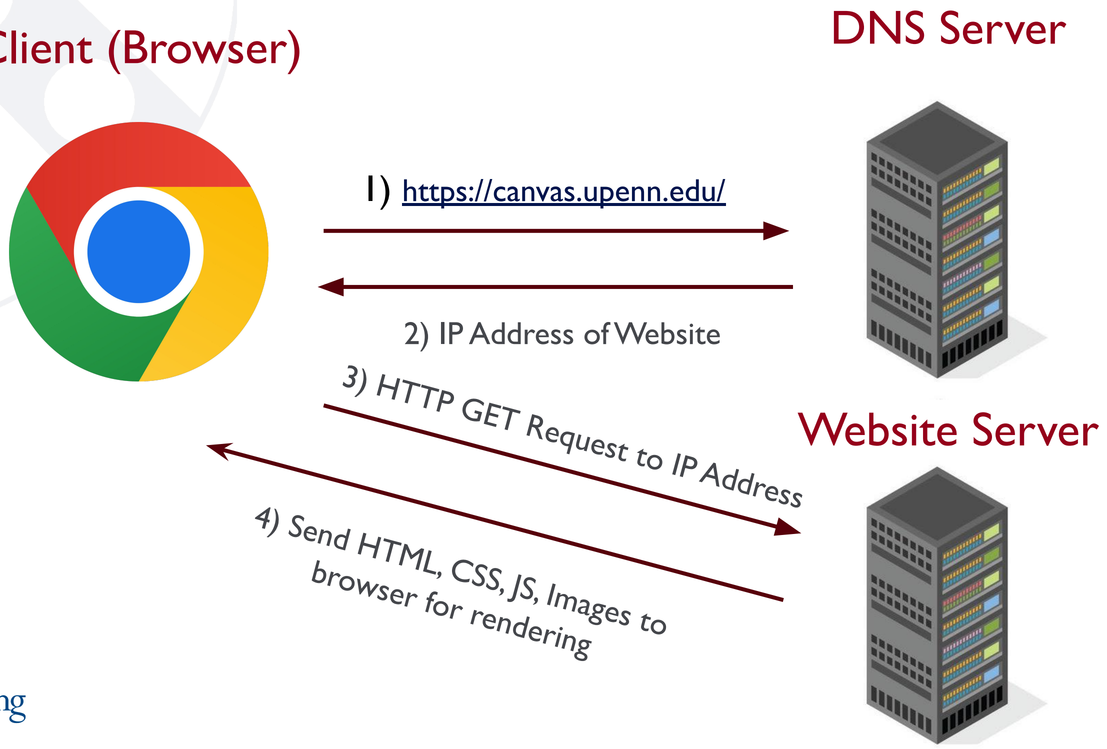
The foundation of all these websites, no matter what new technology arises, remains **HTML, CSS, and JavaScript**.

Modern Web Dev Frameworks

Web development frameworks, like React, Vue, and Angular, are ultimately used to just produce and manipulate HTML, CSS, and JS!

The Client-Server Model

The web functions on a **client-server** model.


Client (Browser)
DNS Server
1) https://canvas.upenn.edu/
2) IP Address of Website
3) HTTP GET Request to IP Address
Website Server
4) Send HTML, CSS, JS, Images to browser for rendering

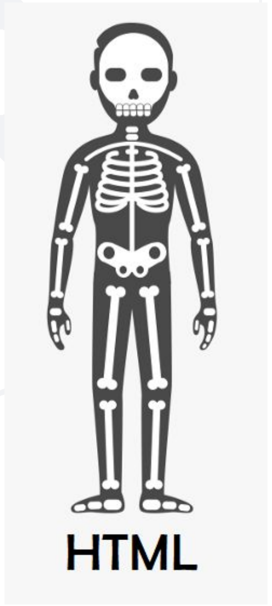
Web Page Rendering

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

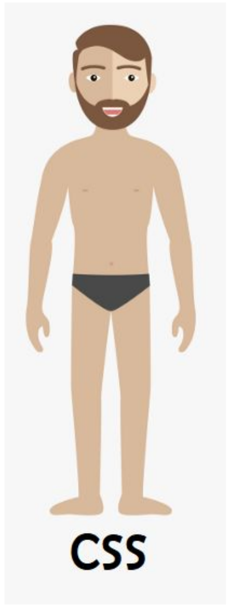
They then apply **CSS** for styling, such as colors, layout, fonts, hovering, and mobile adjustments, among others.

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

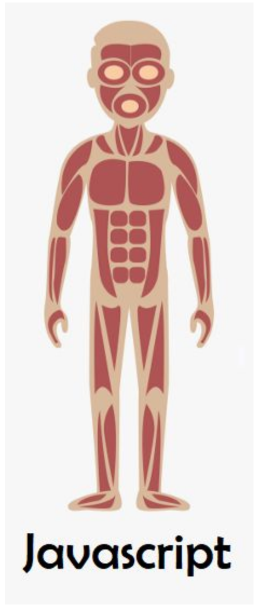
A Web Page is like a human body…



Bones



Skin & Clothes



Muscular/Nervous System

HTML

How do we define the structure of a webpage?

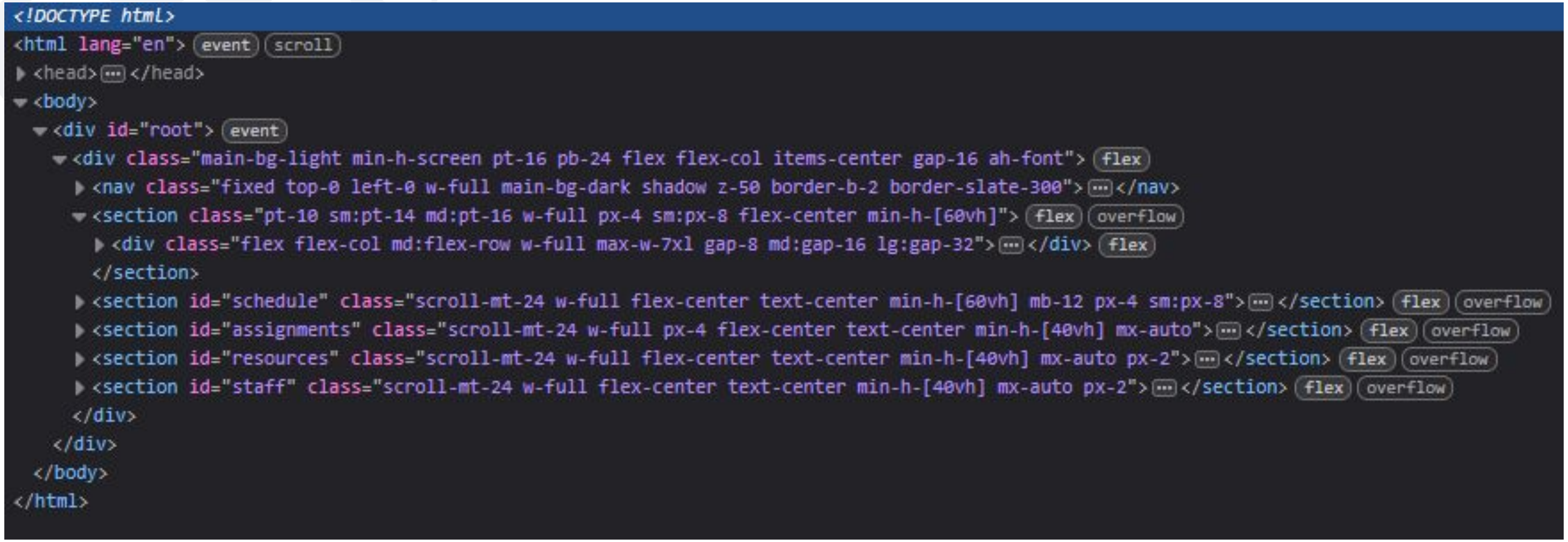
HyperText Markup Language (HTML)

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

Let's study some HTML!

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

Inspect the HTML through Browser Dev Tools:



```
<!DOCTYPE html>
<html lang="en"> event scroll
 ▶ <head> ⋯ </head>
 ▼ <body>
   ▼ <div id="root"> event
     ▼ <div class="main-bg-light min-h-screen pt-16 pb-24 flex flex-col items-center gap-16 ah-font"> flex
       ▶ <nav class="fixed top-0 left-0 w-full main-bg-dark shadow z-50 border-b-2 border-slate-300"> ⋯ </nav>
       ▼ <section class="pt-10 sm:pt-14 md:pt-16 w-full px-4 sm:px-8 flex-center min-h-[60vh]"> flex overflow
         ▶ <div class="flex flex-col md:flex-row w-full max-w-7xl gap-8 md:gap-16 lg:gap-32"> ⋯ </div> flex
         </section>
       ▶ <section id="schedule" class="scroll-mt-24 w-full flex-center text-center min-h-[60vh] mb-12 px-4 sm:px-8"> ⋯ </section> flex overflow
       ▶ <section id="assignments" class="scroll-mt-24 w-full px-4 flex-center text-center min-h-[40vh] mx-auto"> ⋯ </section> flex overflow
       ▶ <section id="resources" class="scroll-mt-24 w-full flex-center text-center min-h-[40vh] mx-auto px-2"> ⋯ </section> flex overflow
       ▶ <section id="staff" class="scroll-mt-24 w-full flex-center text-center min-h-[40vh] mx-auto px-2"> ⋯ </section> flex overflow
       </div>
     </div>
   </body>
</html>
```

(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

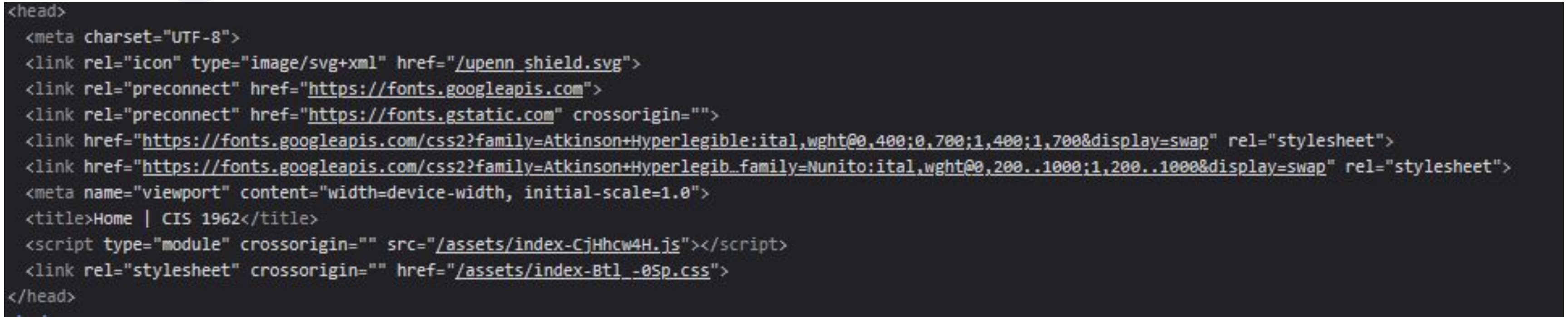
Metadata is the information describing a page. It may include:

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

Class Website Metadata

What metadata does the class website have?

- Links to a favicon image (used in tabs and search engines)
- Links to fonts (Google fonts)
- A defined title
- A stylesheet and script (mostly handled by React)

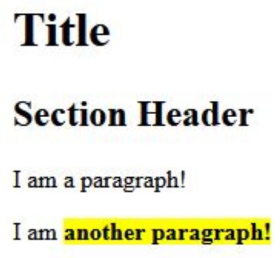


```
<head>
  <meta charset="UTF-8">
  <link rel="icon" type="image/svg+xml" href="/upenn_shield.svg">
  <link rel="preconnect" href="https://fonts.googleapis.com">
  <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin="">
  <link href="https://fonts.googleapis.com/css2?family=Atkinson+Hyperlegible:ital,wght@0,400;0,700;1,400;1,700&display=swap" rel="stylesheet">
  <link href="https://fonts.googleapis.com/css2?family=Atkinson+Hyperlegib…family=Nunito:ital,wght@0,200..1000;1,200..1000&display=swap" rel="stylesheet">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Home | CIS 1962</title>
  <script type="module" crossorigin="" src="/assets/index-CjHhcw4H.js"></script>
  <link rel="stylesheet" crossorigin="" href="/assets/index-Btl_-0Sp.css">
</head>
```

HTML Tags: Grouping & Text

Tag	Description	Example
`<div>`	A generic divider for grouping content and layout	(see below)
`<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>
```


Title

Section Header

I am a paragraph!

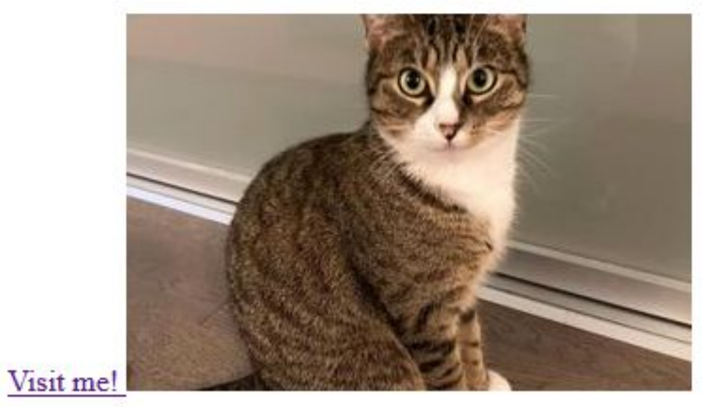
I am **another paragraph!**

HTML Tags: Links & Images

Tag	Description	Example
`<a>`	Anchor, defines hyperlinks with href attribute	`<a href="https://example.com"> Visit me! </a>`
`<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.	`<img src="https://placecats.com/300/200" alt="A cute cat">`

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

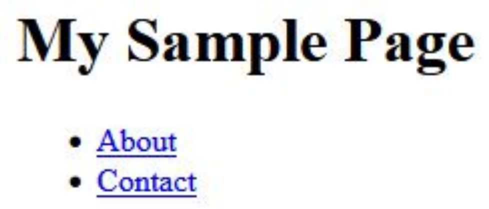
```
<a href="https://example.com">
   Visit me!
</a>
<img
src="https://placecats.com/300/200"
alt="A cute cat">
```



HTML Tags: Lists and Navigation

Tag	Description	Example
`<ul>`,`<ol>`,`<li>`	Unordered/ordered lists and list items	
`<nav>`	Navigation section, for site menus	
`<header>`, `<footer>`	The top and bottom sections of a site	

```
<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>
```



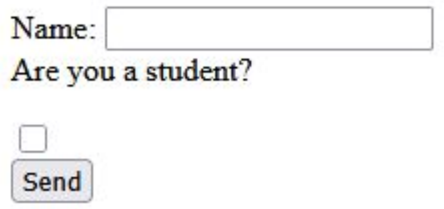
My Sample Page

- About
- Contact

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	

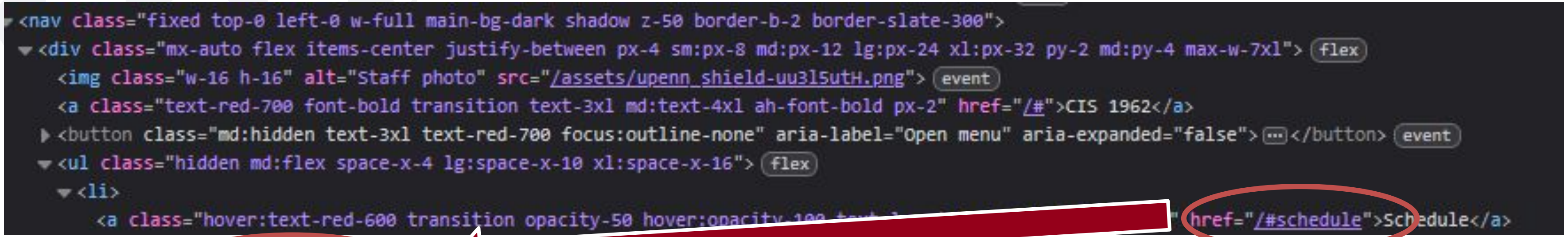
```html
<form>
   <div>
      <label for="name">Name:</label>
      <input type="text" name="name"/>
   </div>
   <div>
      <label for="checkbox">Are you a
student?</p>
      <input type="checkbox"
name="checkbox"/>
   </div>
   <button type="submit">Send</button>
</form>
```



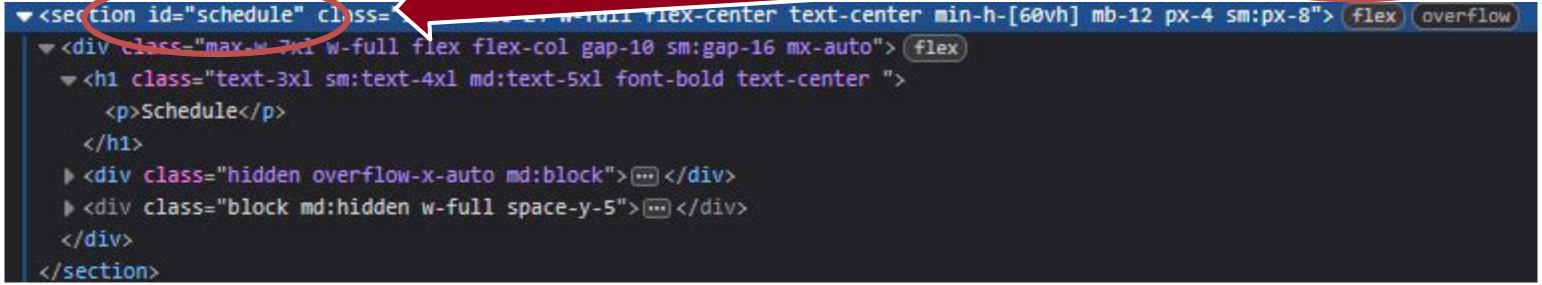
Attributes: href and id

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.



```
▾<nav class="fixed top-0 left-0 w-full main-bg-dark shadow z-50 border-b-2 border-slate-300">
  ▾<div class="mx-auto flex items-center justify-between px-4 sm:px-8 md:px-12 lg:px-24 xl:px-32 py-2 md:py-4 max-w-7xl"> flex
     <img class="w-16 h-16" alt="Staff photo" src="/assets/upenn_shield-uu315utH.png"> event
     <a class="text-red-700 font-bold transition text-3xl md:text-4xl ah-font-bold px-2" href="/#">CIS 1962</a>
    ▸<button class="md:hidden text-3xl text-red-700 focus:outline-none" aria-label="Open menu" aria-expanded="false"> ⋯ </button> event
    ▾<ul class="hidden md:flex space-x-4 lg:space-x-10 xl:space-x-16"> flex
      ▾<li>
         <a class="hover:text-red-600 transition opacity-50 hover:opacity-100 text-l                    " href="/#schedule">Schedule</a>
```



```
▾<section id="schedule" class="                   -full flex-center text-center min-h-[60vh] mb-12 px-4 sm:px-8"> flex overflow
  ▾<div class="max-w-7xl w-full flex flex-col gap-10 sm:gap-16 mx-auto"> flex
    ▾<h1 class="text-3xl sm:text-4xl md:text-5xl font-bold text-center ">
        <p>Schedule</p>
     </h1>
    ▸<div class="hidden overflow-x-auto md:block"> ⋯ </div>
    ▸<div class="block md:hidden w-full space-y-5"> ⋯ </div>
   </div>
 </section>
```

Semantic & Non-Semantic Tags

Non-semantic tags, such as <div> and <span> say nothing about their content.

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

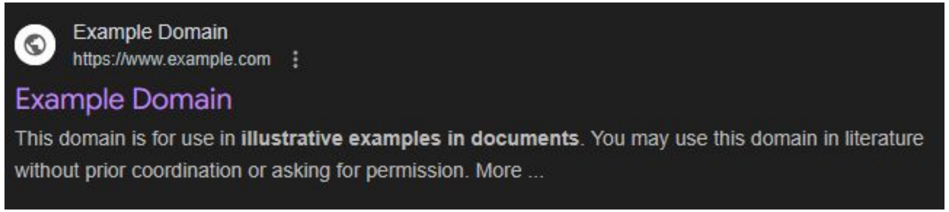
```
<div class="big"> Neo the Cat </div>
<div> Neo is a cute cat! </div>
<div>
     <img
src="https://placecats.com/neo/300/200">
</div>
```

```
<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 Domain
https://www.example.com

Example Domain

This domain is for use in **illustrative examples in documents**. You may use this domain in literature without prior coordination or asking for permission. More ...

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>
```

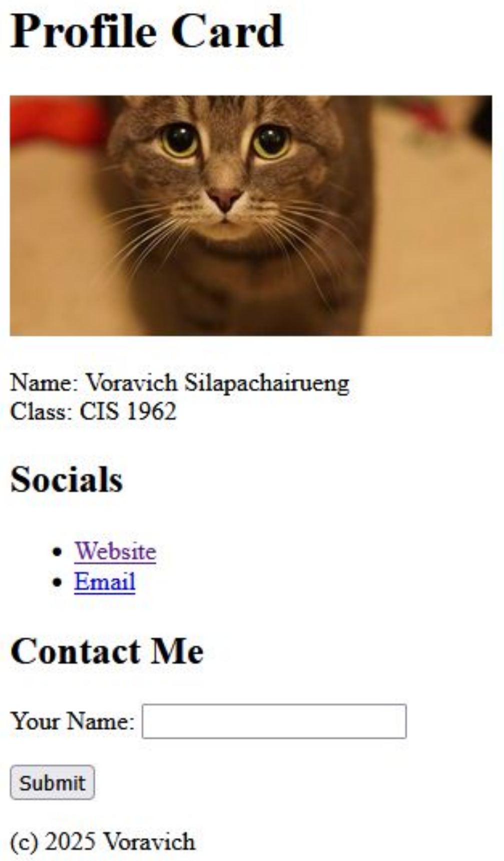
```
<html>
  └── <body>
        └── <section>
              ├── <h2>
              └── <ul>
                    └── <li>
```

Live Coding Activity: Make this Site!

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!



Profile Card

Name: Voravich Silapachairueng
Class: CIS 1962

Socials

- Website
- Email

Contact Me

Your Name:

Submit

(c) 2025 Voravich

CSS

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

Penn Engineering

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

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

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

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

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

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

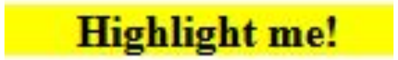
Style any <li> elements inside an <ol> element to be italicized.

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;
}
```



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

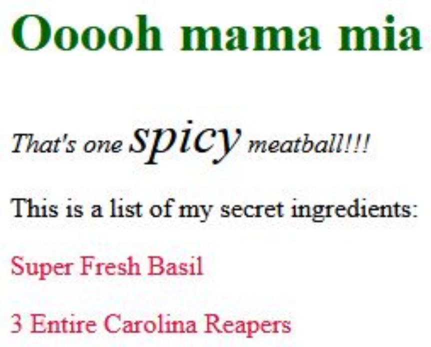
```css
h1 {
  color: darkgreen;
}

#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>
```


Ooooh mama mia

That's one spicy meatball!!!

This is a list of my secret ingredients:

Super Fresh Basil

3 Entire Carolina Reapers

Cascading Styles

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

Three main principles, ordered by what applies with 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.

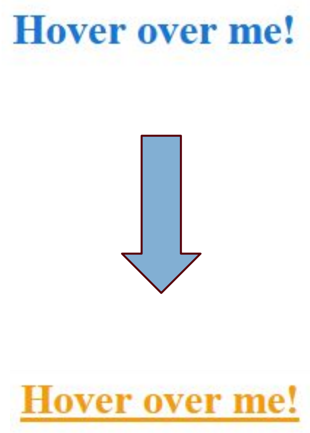
For instance:

- `: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

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

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



Important CSS Properties

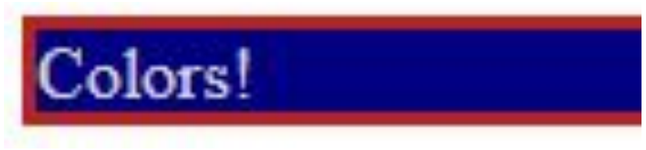
- **Colors:** `color, background-color`
- **Borders:** `border, border-radius`
- **Text:** `font-size, font-weight, font-family, text-align`
- **Spacing:** `margin, padding`
- **Layout:** `display, width, height, justify-content, align-items, flex-direction`

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;
}
```

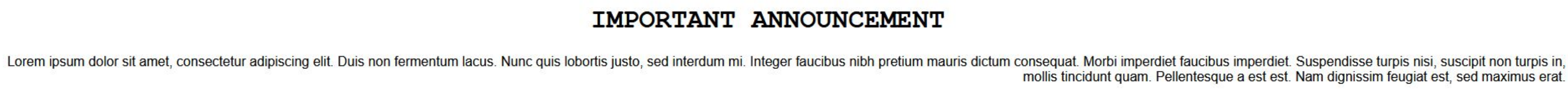
Penn Engineering

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.

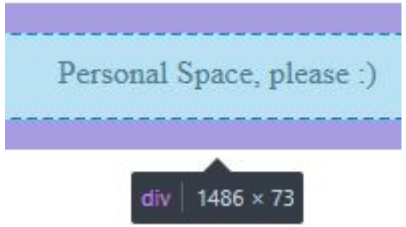
CSS Properties: Spacing

Spacing: `margin, padding`

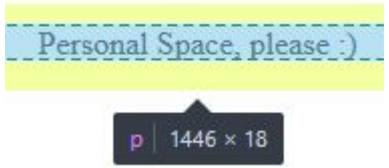
You can manipulate the margin/padding in all 4 directions.

```
div {
     padding: 15px;
}

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



margin



padding

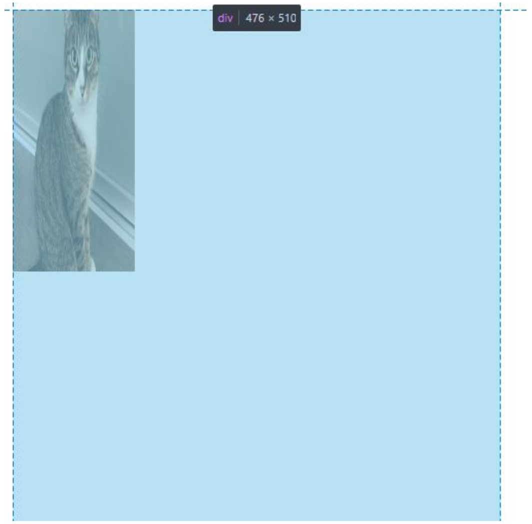
CSS Properties: Width/Height Layout

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;
}
```

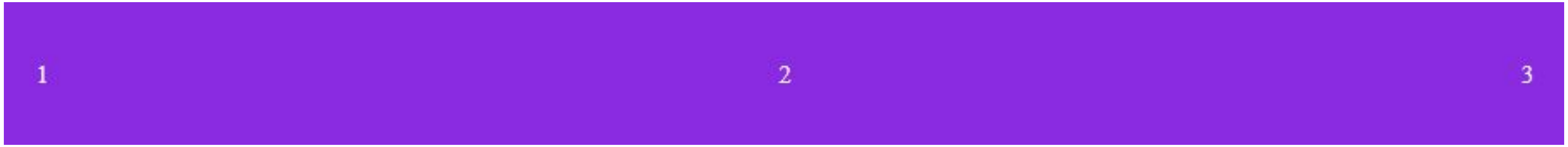


CSS Properties: Display & Flexbox

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
}
```



Honorable Mention: Flexbox Froggy

https://flexboxfroggy.com/

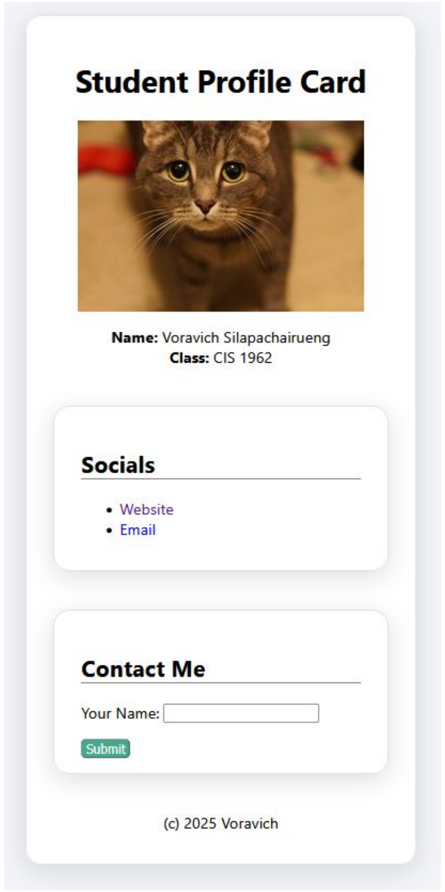
This site will give you great practice with

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?
- Don't forget: add your stylesheet to the header!



Student Profile Card

Name: Voravich Silapachairueng
Class: CIS 1962

Socials

- Website
- Email

Contact Me

Your Name:

(c) 2025 Voravich

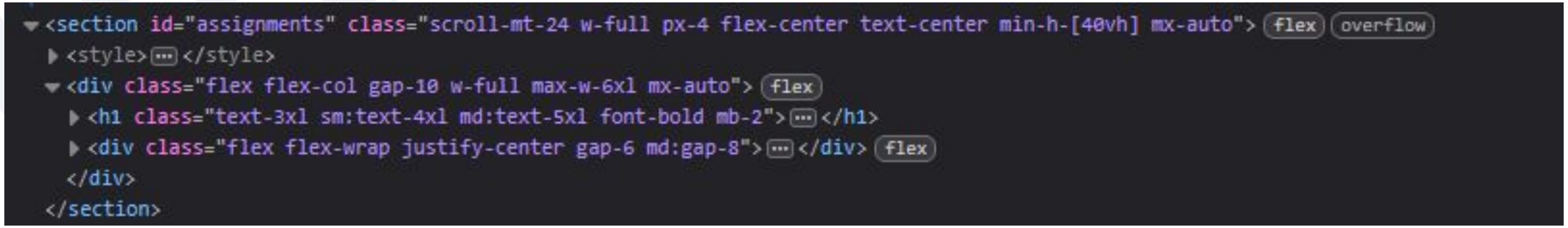
CSS Frameworks

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

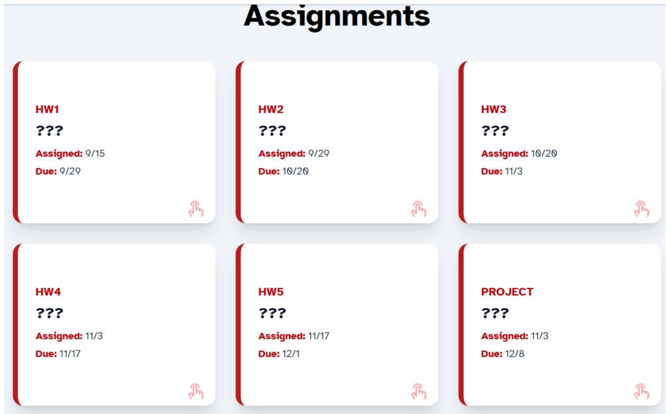
These are CSS frameworks such as **Bootstrap, Bulma, Tailwind, and Foundation**. Most include pre-written classes that can be applied to unify the look of an app quickly.

CSS Frameworks Example: Tailwind

Our class website uses Tailwind! Take a look at how Tailwind's pre-written classes get applied to a section of the web page:



```
<section id="assignments" class="scroll-mt-24 w-full px-4 flex-center text-center min-h-[40vh] mx-auto"> flex overflow
  <style>…</style>
  <div class="flex flex-col gap-10 w-full max-w-6xl mx-auto"> flex
    <h1 class="text-3xl sm:text-4xl md:text-5xl font-bold mb-2">…</h1>
    <div class="flex flex-wrap justify-center gap-6 md:gap-8">…</div> flex
  </div>
</section>
```



Assignments

HW1
???
Assigned: 9/15
Due: 9/29

HW2
???
Assigned: 9/29
Due: 10/20

HW3
???
Assigned: 10/20
Due: 11/3

HW4
???
Assigned: 11/3
Due: 11/17

HW5
???
Assigned: 11/17
Due: 12/1

PROJECT
???
Assigned: 11/3
Due: 12/8

5-Minute Break!

DOM Manipulation

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

Penn Engineering

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

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

Hello Everyone!

```javascript
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>
```

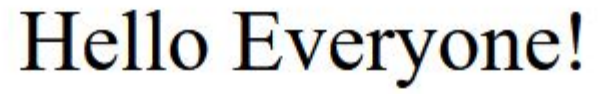
You can also attach external scripts:

```
<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>
```



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

Selecting Elements By Tag/Class

You can also select by tags or classes:

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

```javascript
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.

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>
```

textContent

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

innerHTML

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

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.

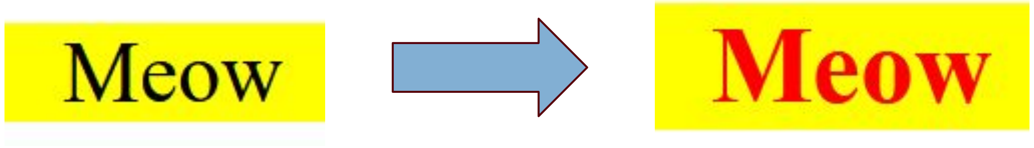
`element.classList`: Includes various methods to add CSS classes to the `class` attribute, including:

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

Changing Style Example

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

```javascript
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);
```

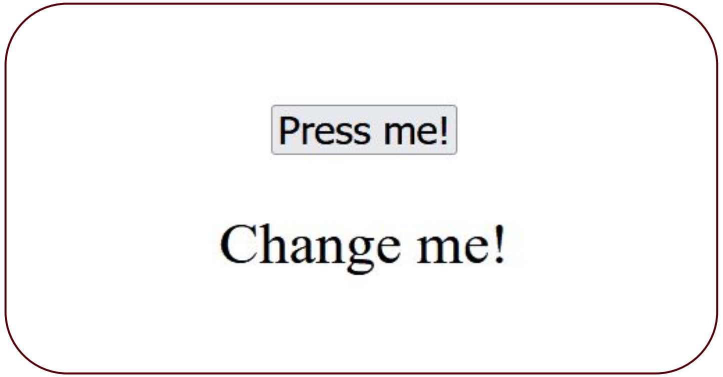
element: HTML element that this listener waits on an event for

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 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")
     })
```

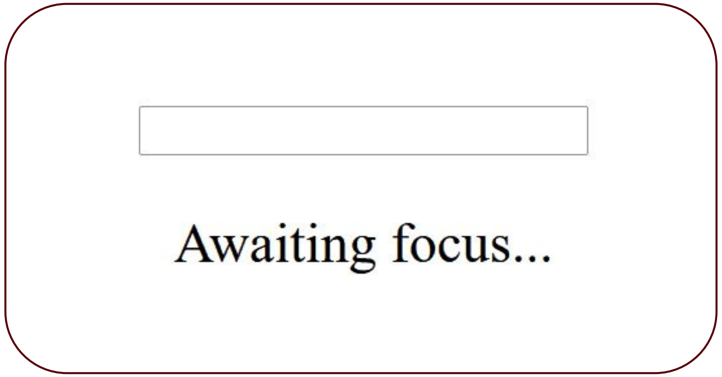
```
document.getElementById("demo-btn")
     .addEventListener('mouseleave', () => {
       document.getElementById("demo-btn")
          .classList.toggle("fancy-button")
     })
```

Penn Engineering

Event Types: Focus (In/Out)

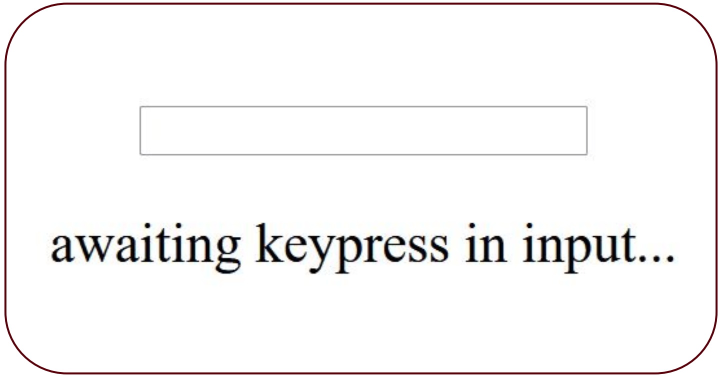
```
document.getElementById("focus-input")
    .addEventListener('focusin', (e) => {
       document.getElementById("focus")
         .textContent = `Focusing on input!`
    })
```

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


Awaiting focus...

Event Types: Typing (‘keydown’)

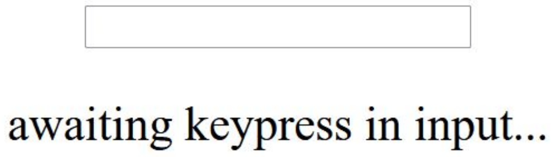
```
document.getElementById("await-input")
    .addEventListener('keydown', (e) => {
        document.getElementById("await")
         .textContent = `You typed ${e.code}`
    })
```


awaiting keypress in input...

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?


awaiting keypress in input...

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

event.preventDefault()

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

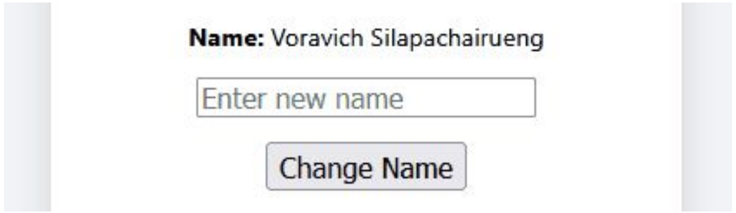
You can prevent this using event.preventDefault().

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

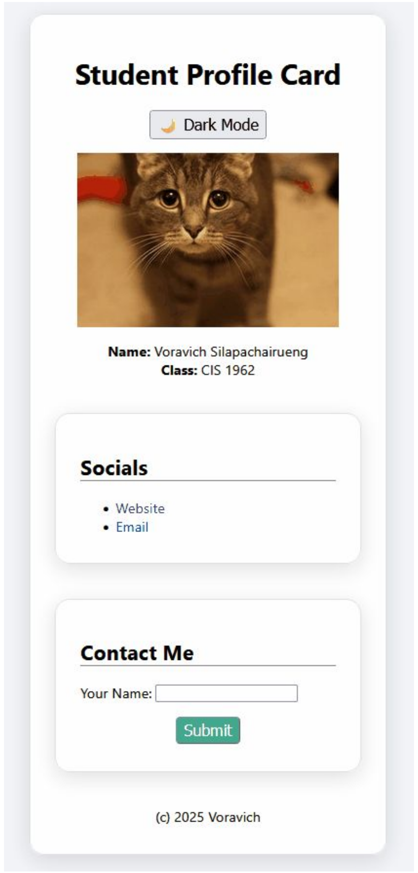
Live Coding Activity: Make it Interactive!

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!



Name: Voravich Silapachairueng



Student Profile Card

Name: Voravich Silapachairueng
Class: CIS 1962

Socials

- Website
- Email

Contact Me

Your Name:

(c) 2025 Voravich

Penn Engineering