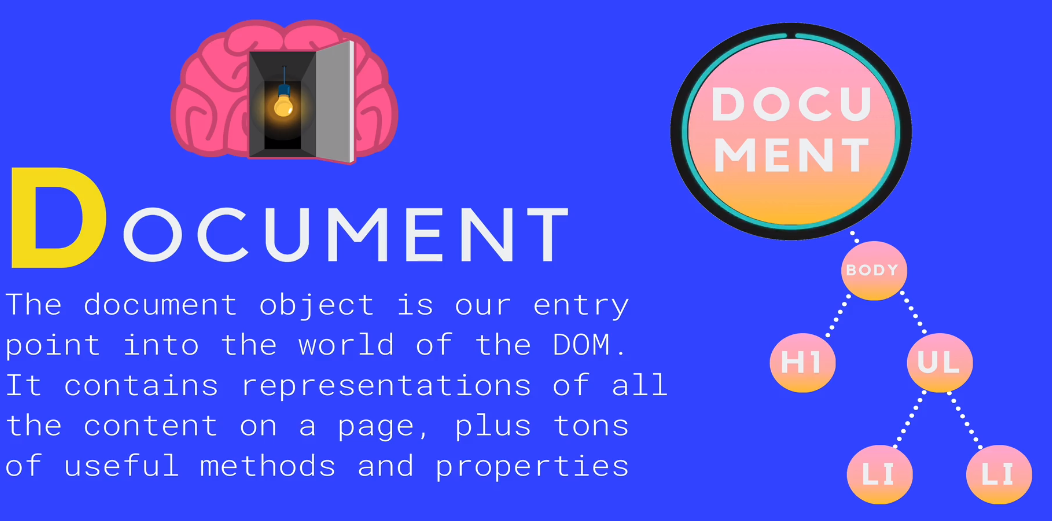
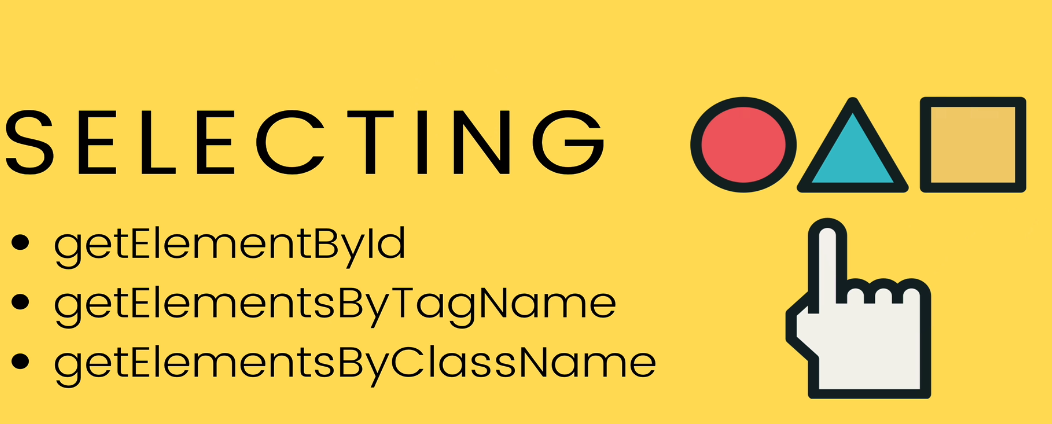
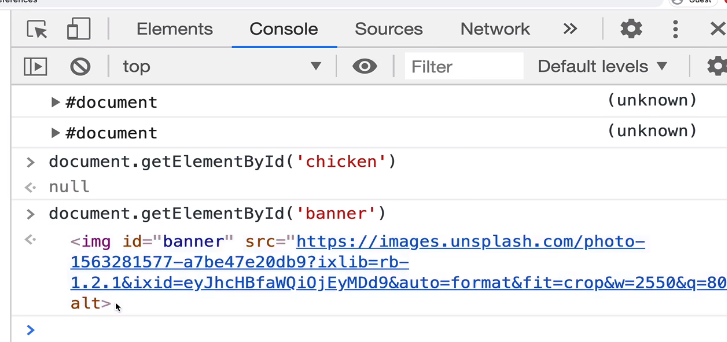
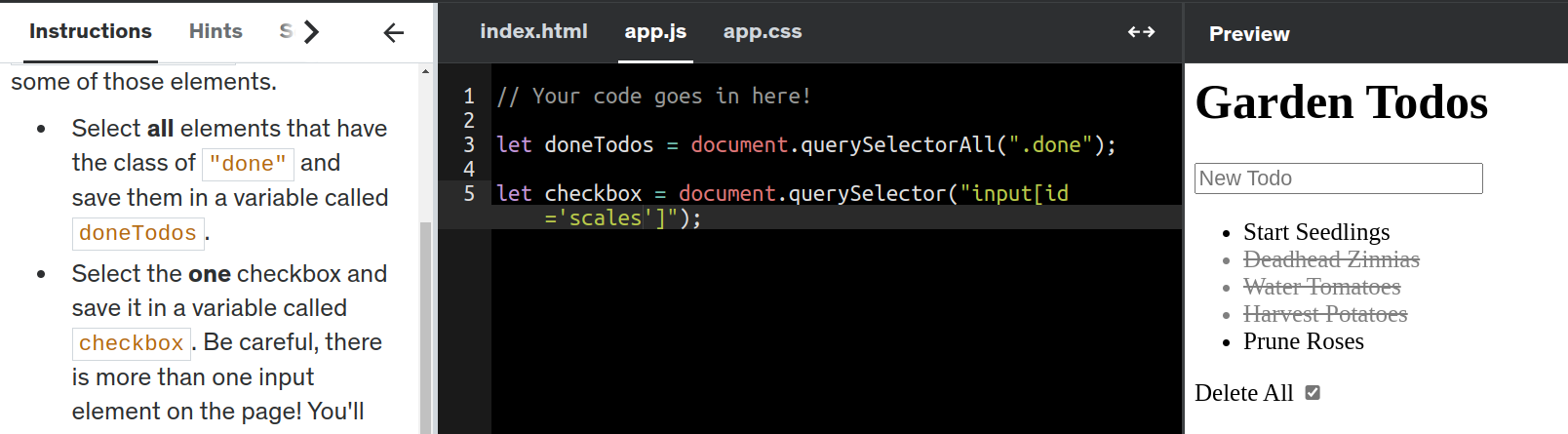
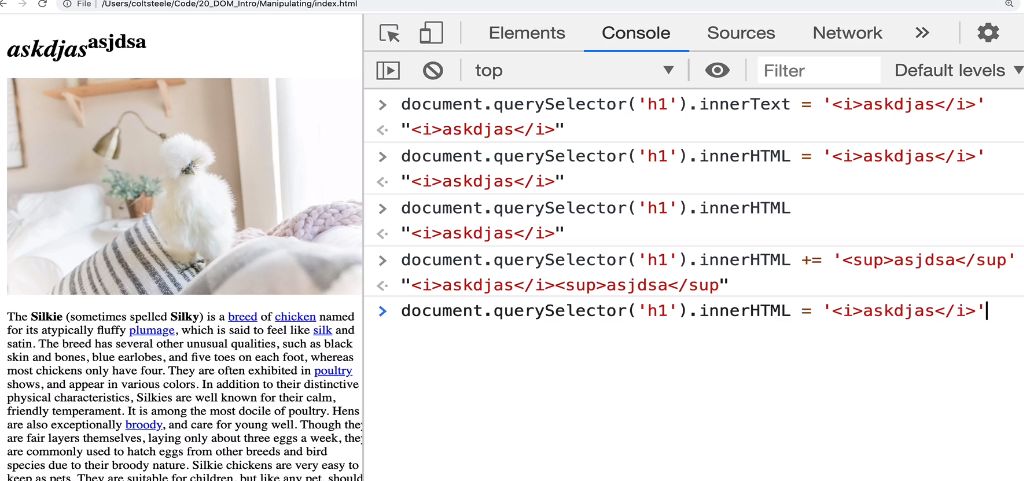
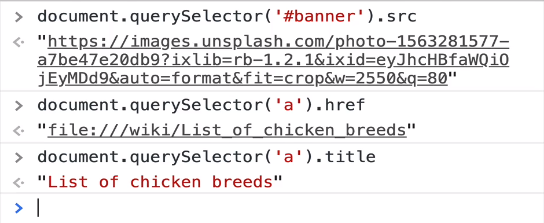
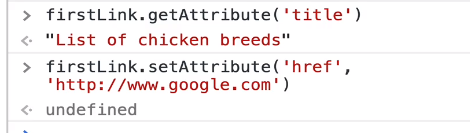
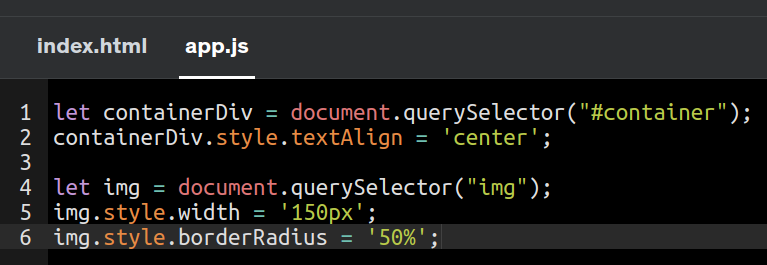
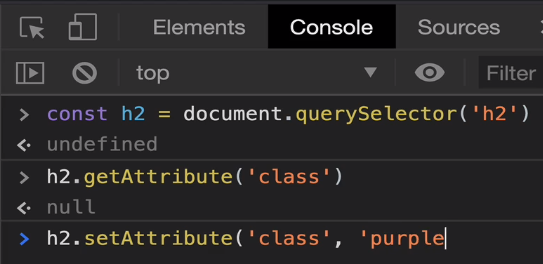
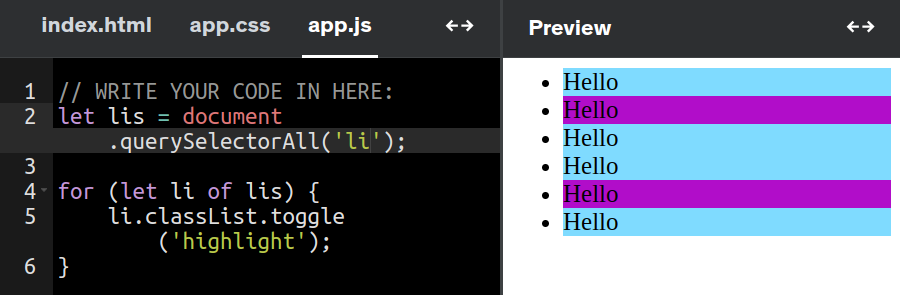
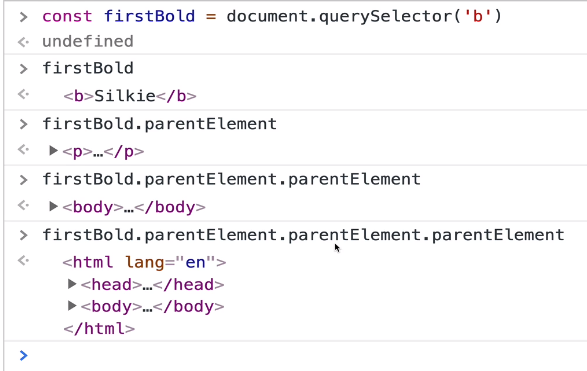
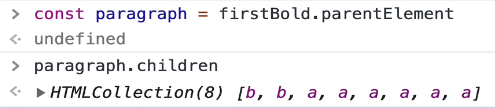
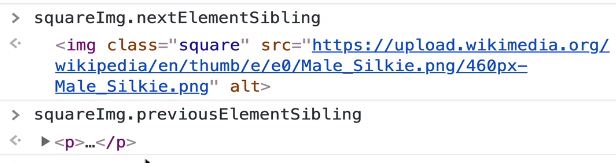
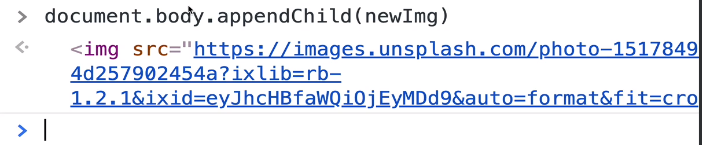
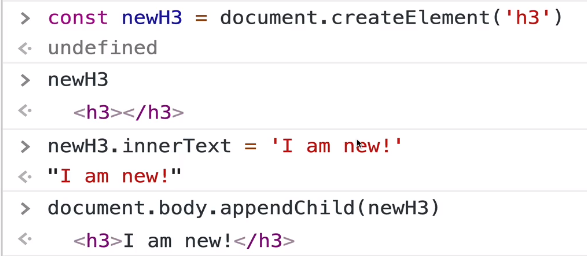
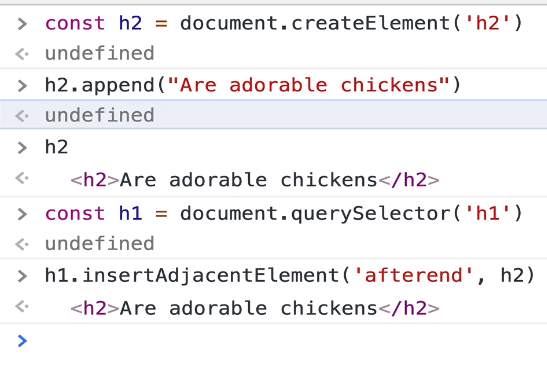
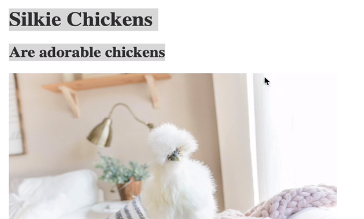
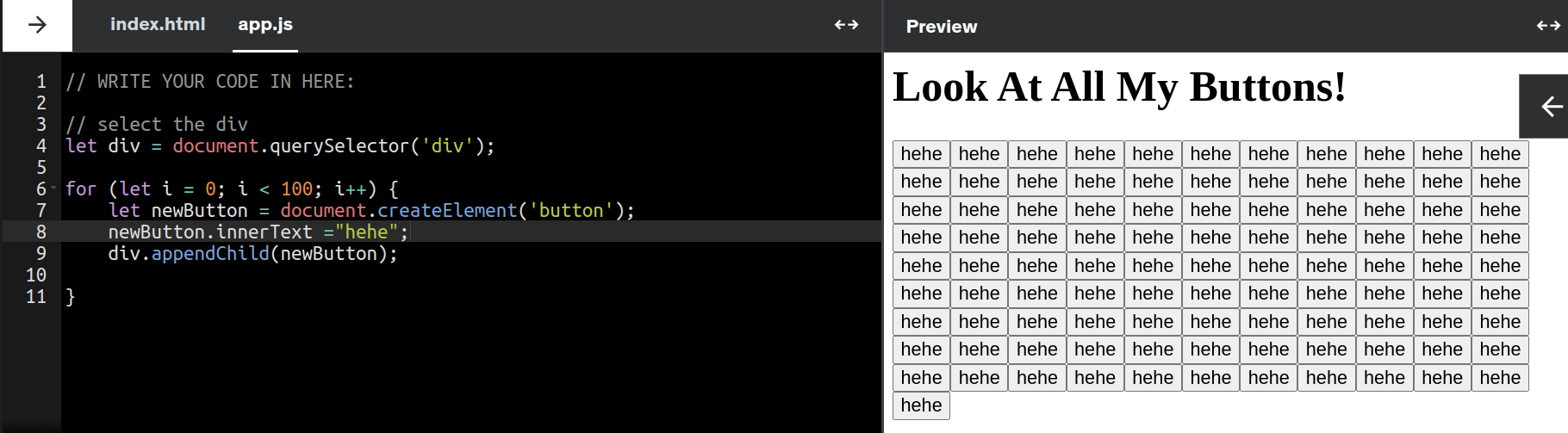
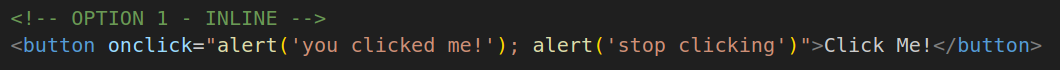
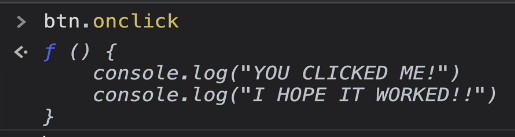
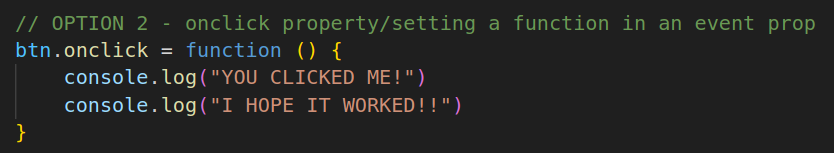
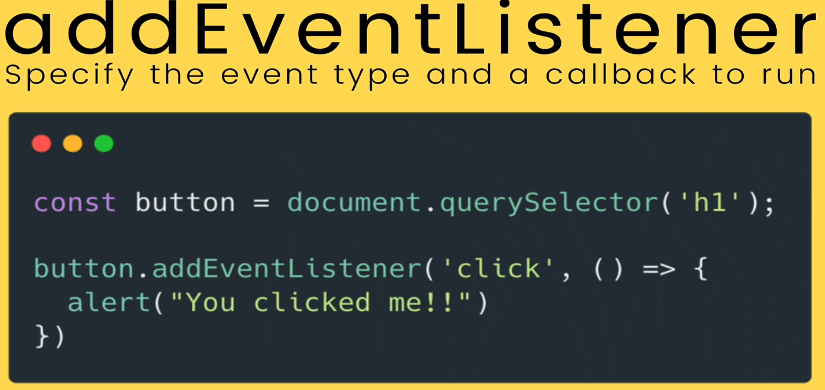
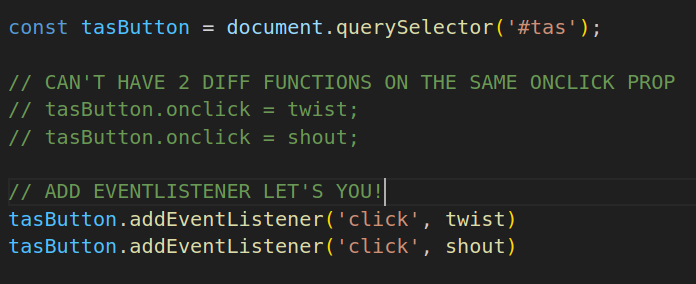
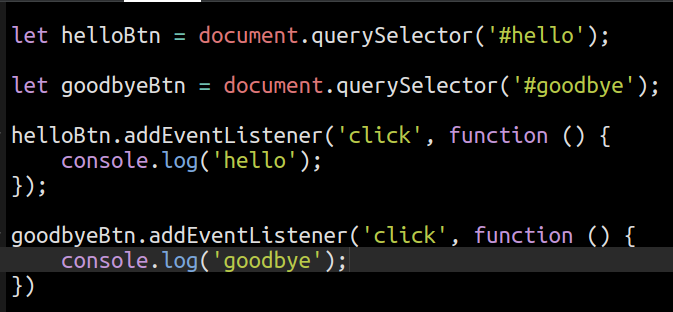
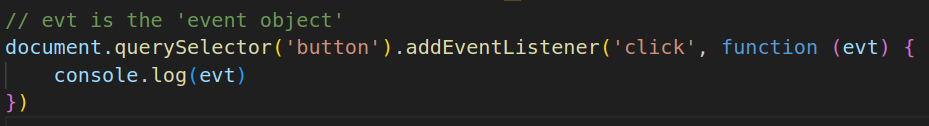
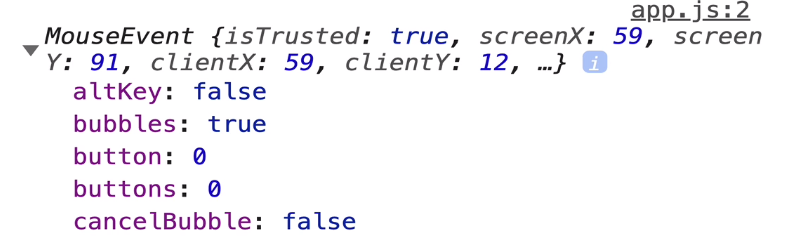
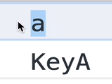
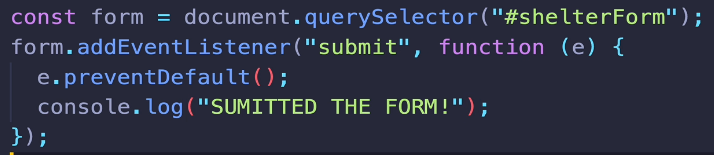
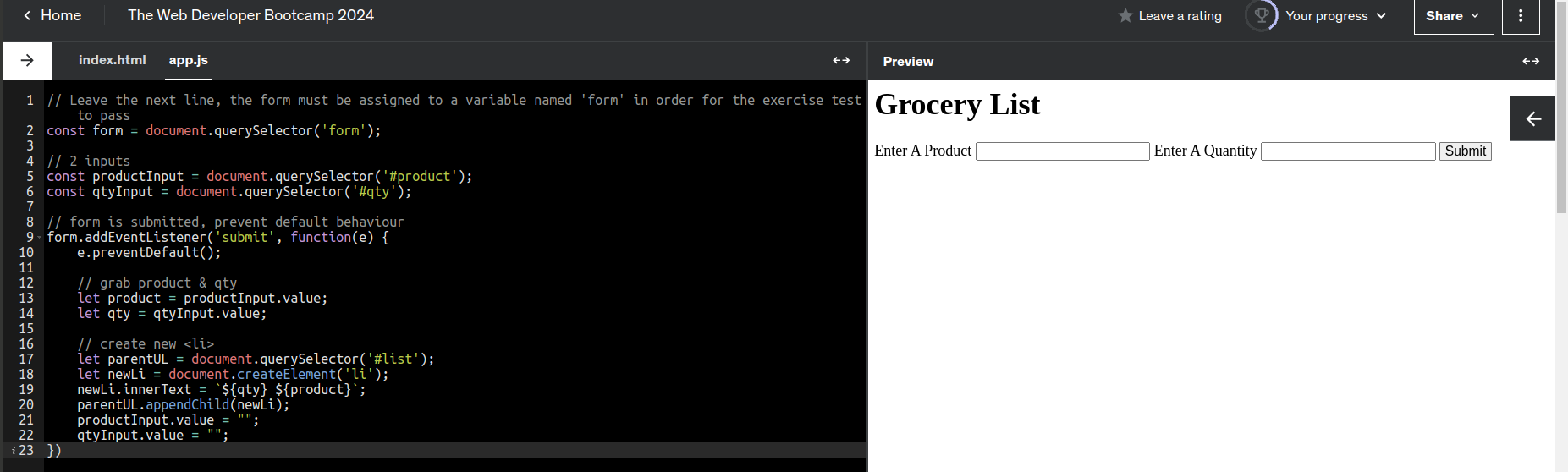
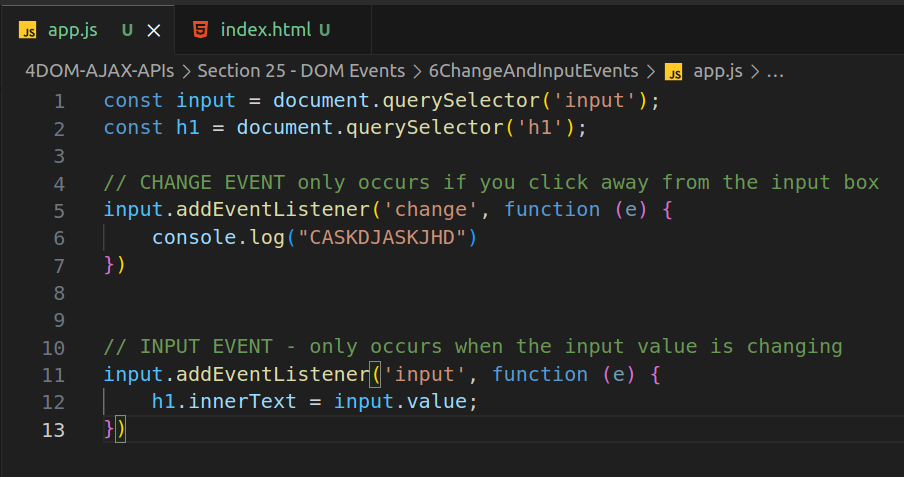
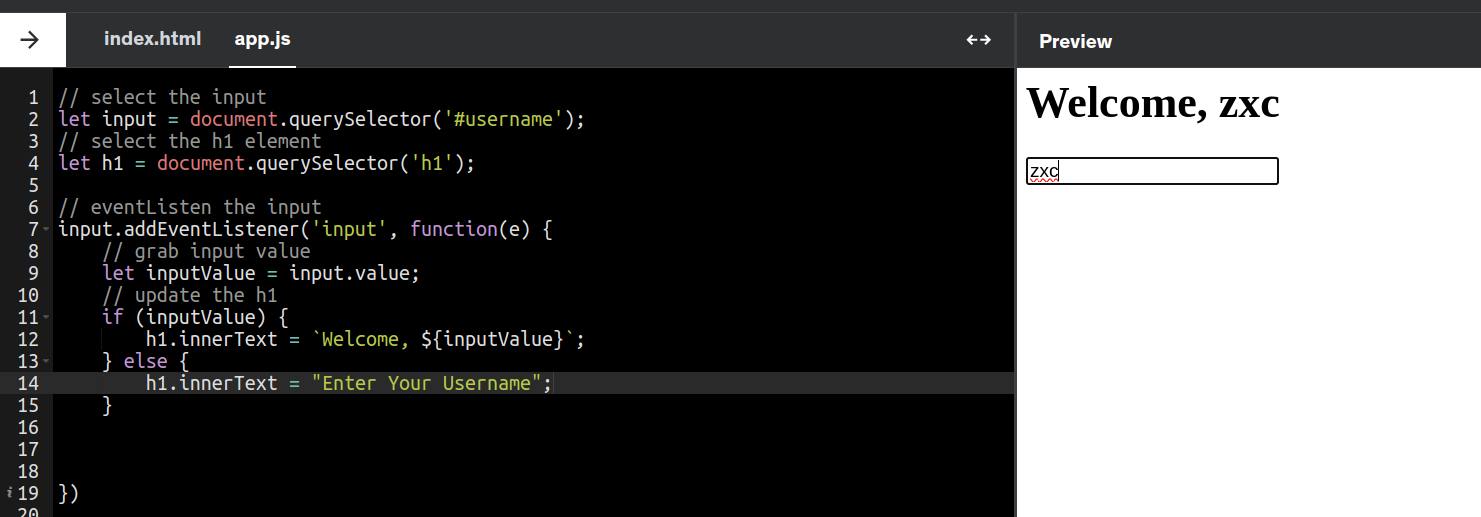
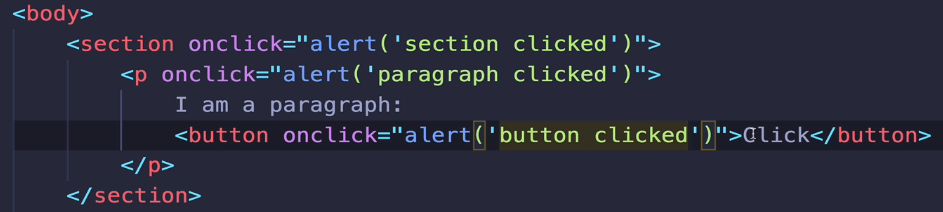
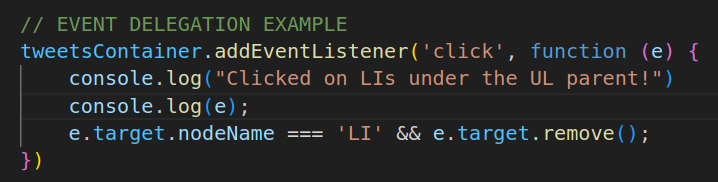
Web Dev Notes - DOM

# Section 24 – Intro to DOM

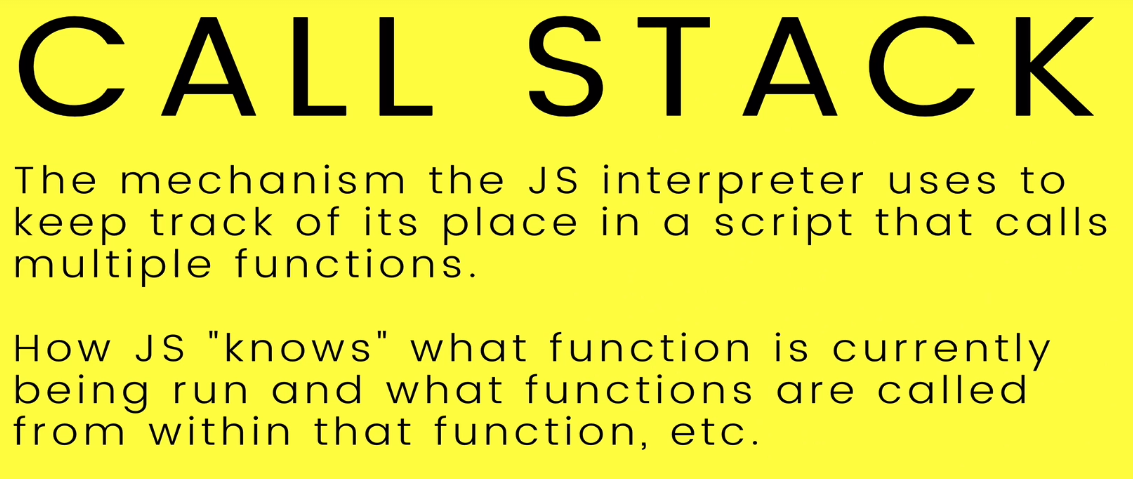
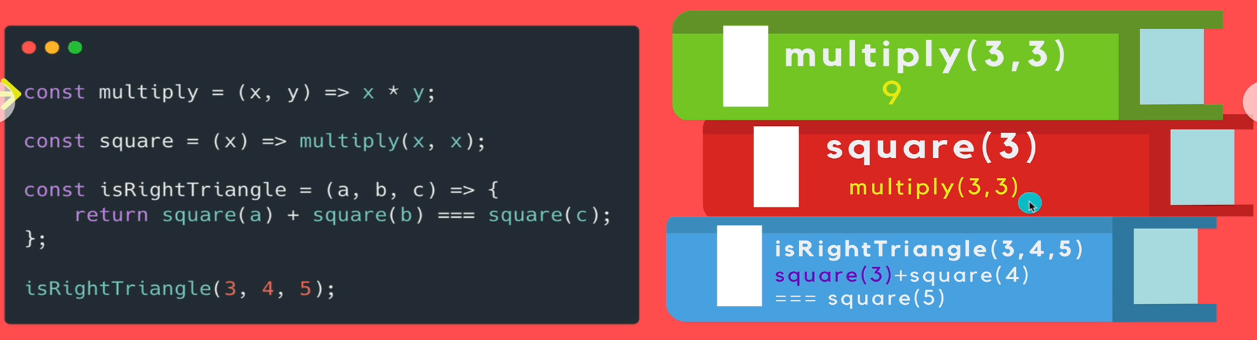
* **Document Object**
  + This represents the page itself. It is a JS object
  + 
  + HTML and CSS components are converted into JS objects
* **SELECTING** methods are used to choose an element in your document
  + 
  + Example of **getElementById**
    - 
  + Example of **getElementByTagName**
    - Rebecca Donaldson
* **SELECTING Modernized:**
  + **QuerySelector()**
    - 
      * Just gives us the first matching element
  + **QuerySelectorAll()**
    - Returns a **collection** of matching elements
      * 
* **DOM MANIPULATION**
  + Methods and Properties of DOM
    - We're working hard to fix it, so please try again soon.
    - **Example of innerText, innerHTML and textContent**
      * 
      * let updatedText = document.querySelector("h1 span").innerHTML = "<b>Disgusting<b>"
* **ATTRIBUTES**
  + Example of *src* attribute
    - 
  + you can also use **getAttribute()** or **setAttribute()**
    - 
      * **Example**:
        + let source = document.querySelector('img');
        + source.setAttribute('src', 'https://devsprouthosting.com/images/chicken.jpg');
        + source.setAttribute('alt', 'chicken');
* **CHANGING STYLES**
  + <html tag>.**style**
    - **ex.** h1.style
    - Contains ALL props of CSS that a HTML tag can use
    - This style object DOESN’T contain the styles coming from a css style sheet (**ex**. app.css)
    - Instead, the style object is checking the ‘inline’ style settings in your HTML tag
      * **ex**. <h1 style=”color": blue”>blah</h1>
    - You can also use **window.getComputedStyle(h1)** to get all the css values per tag
    - 
    - 
* **CLASSLIST**
  + **Option 1**: You can retrieve the class property of an element and then set it’s attribute/prop to a specific value
    - ex. 
  + **BETTER option**: Use .classList to retrieve all classes under an element
    - 
      * Using .**add** or .**remove** would change the class value in the tag
      * You can also use .**toggle** as it does those 2 operations
        + 
        + 
* **Traversing the DOM**
  + How to search through elements in the DOM from BOTTOM to TOP )
    - .parentElement
      * 
  + You can also search to see ALL the children underneath an element
    - * .children
      * 
  + You can also search by adjacent sibling (.nextElementSibling)
    - 
* **AppendChild**
  + **Steps**:
    - Create new elements and adding it to other elements
      * const newImg = document.createElement(‘<tag>’)
        + Ex. newImg.src = ‘<image location>’
    - AppendChild to parent tag
      * **Ex**. Append to body tag
        + 
      * **Ex2**. Append a new h3 to body
        + 
* **Append –** Similar, to appendChild but more flexible!
  + ParentNode.append() --> adds to end of node
    - You could append additional text into/inside a <p> tag
      * 
    - You could append multiple tags
  + .**prepend**() --> adds to beginning as first child of element
* .**insertAdjacentElement** --> Insert between elements
  +  **=** 
  + Inserted as next sibling, not as a child
    - 
  + Exercise: Append 100 buttons to div as a child
    - 
* **.removeChild**()
  + Removes the child from an element, not the element itself
    - 
    - 
* .**Remove**() --> will remove the element it is called on
  + 

# Section 25 – DOM Events

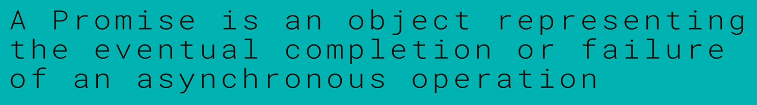
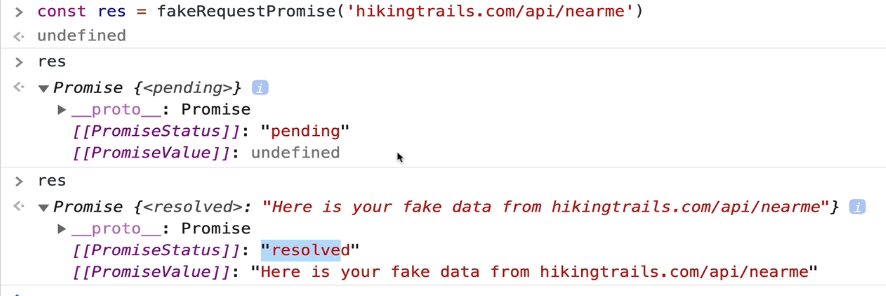
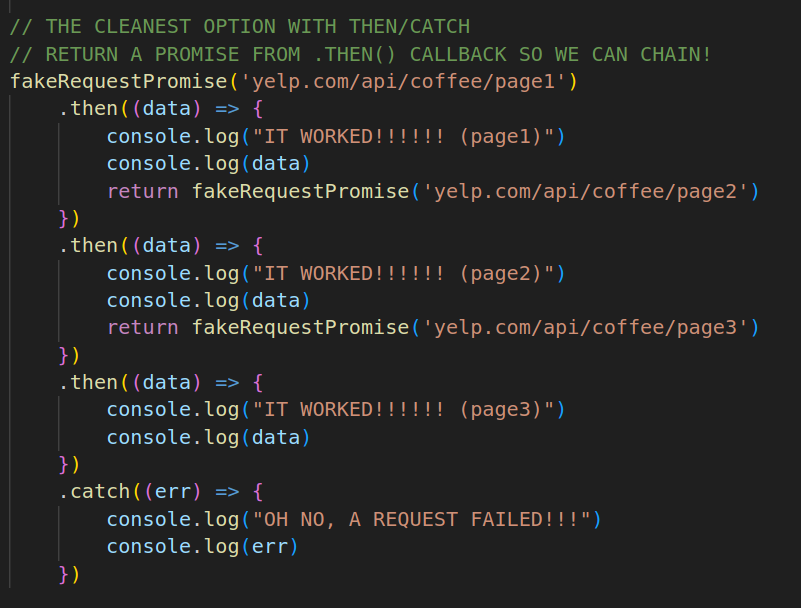
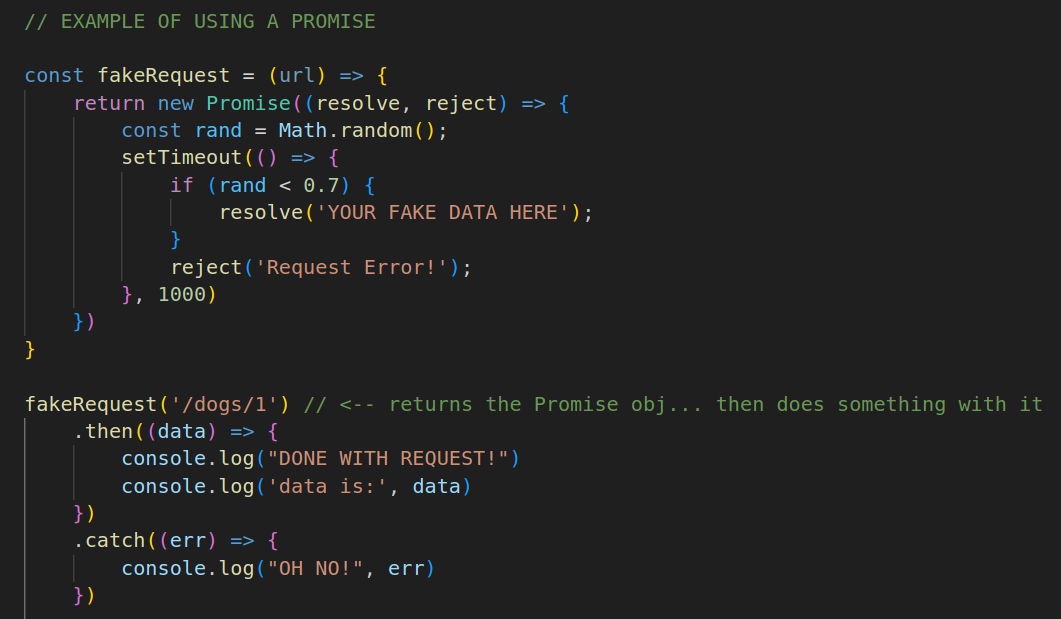
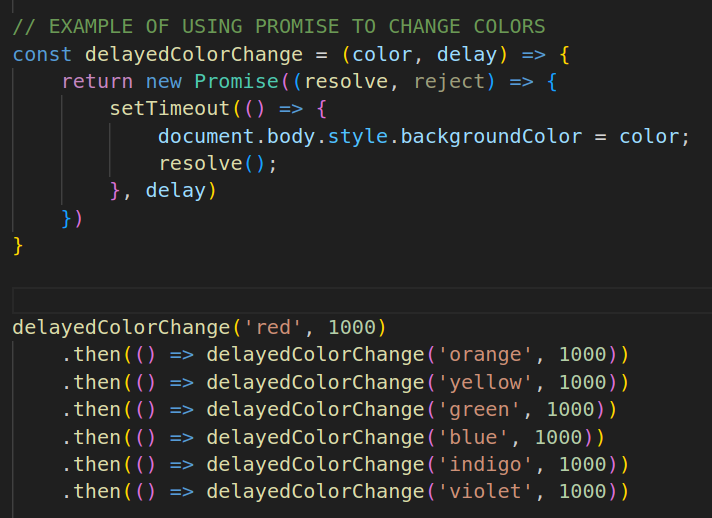
* **Events = response to user input**
  + Some events:
    - 
* **3 Types of Events to manage User Input**
  + 1) Inline Events (not recc)
    - use onClick:
      * 
  + 2) Onclick Property
    - Set this property as a function
      * 
      * First in app.js, select the html element you want to alter the property for
        + Then add a function as a property of the element
        + 
  + 3) addEventListener() (*best approach!*)
    - 
    - Takes 2 args: 1) event type; 2) callback function
    - This method works better because in option 2, you can only set 1 value for the property at a time. If you have 2 functions executing onClick, only 1 will run
    - addEventListener lets you run multiple functions
      * 
    - **Example**:
      * 
* **Events and THIS**
  + 
  + this --> references the object you are calling the function on
* **Event Object**
  + 
    - This object (evt or e)represents an event that’s occurred example: ‘mouseEvent’
    - IT is auto passed in by the event handler
      * 
    - Consider these 2 properties in the KeyEvent:
      * console.log(e.key) --> the actual character that was pressed/actual result
      * console.log(e.code) --> the name of the key (location of key on keyboard)
      * **Output**:
        + 
* **Form Events & PreventDefault()**
  + Form ‘actions’ send you to a new page
  + e.PreventDefault()
    - It prevents the default behaviour triggered by an event
    - 
  + **Example**:
    - 
* **Change Events and Input Events**
  + 
    - change event --> occurs when you enter a value and then click away from box
    - input event --> occurs when the input value changes
    - **Example**
      * 
* **Event Bubbling**
  + Clicking on an element that is a child of other elements will cause the events to ‘bubble up’ aka trigger upwards
    - Clicking the <button>, triggers <p>, then <section>
      * 
    - e.stopPropagation() --> stops the bubbling
      * 
* **Event Delegation**
  + Newly created elements by .addEventListener don’t inherit the event listener (ex. New <li> won’t inherit the event listener that
  + Instead, use event delegation by adding event listener to the parent element. The child elements (newly created/already exist) will also get the same event listener
  + Example
    - 
    - .target --> indicates the specific element clicked (ex. <li> under <ul> parent)

# Section 27 – Async

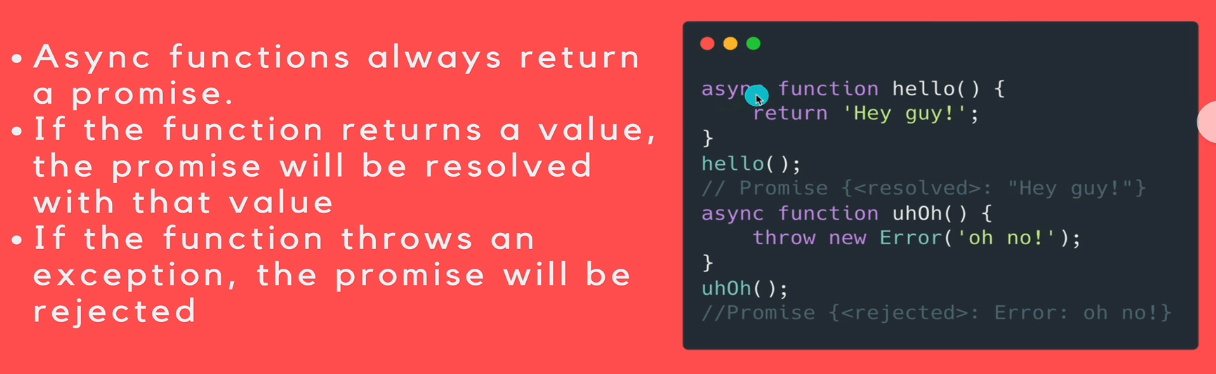
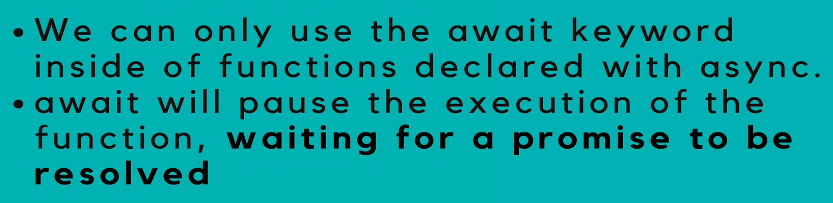
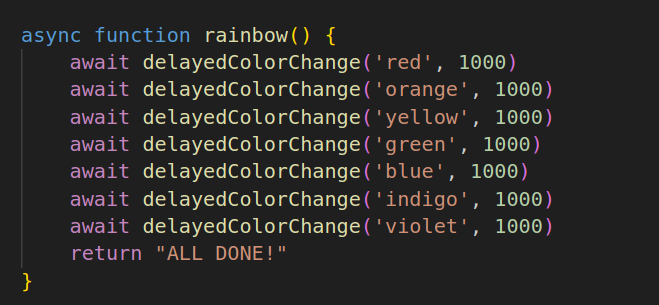
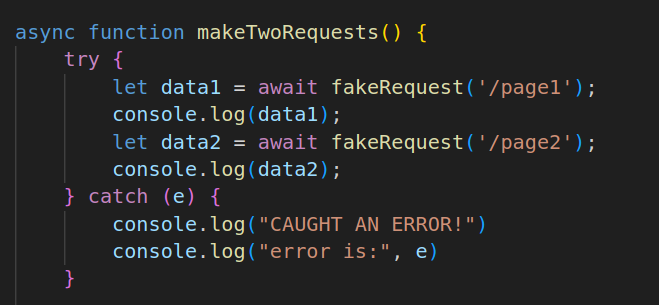
## Call Stack

* 
  + **Example**: *How a stack of functions operate (1 Function calls 2 others)*
    - 
      * Functions are called in order of top to bottom, left to right
        + aka. LIFO
    - JS is single-threaded

## PROMISES

* 
  + **Example of using Promise:**
    - 
      * It produces a ‘Promise {<resolved>}’
        + Promise object can have 3 states: *pending, resolved, rejected*
    - 
  + **Two Methods of writing Promises:**
    - 1) ***Call*** *the promise within the .then() of the previous promise* (**ex**. Call page 1, then page 2)
      * 
    - 2) ***Return*** *the next promise within the .then() of the previous promise*
      * 
  + **EX 1**: *Return data based on resolved or rejected*
    - 
  + **EX 2**: *Sequential Colour Change*
    - 

## ASYNC FUNCTIONS

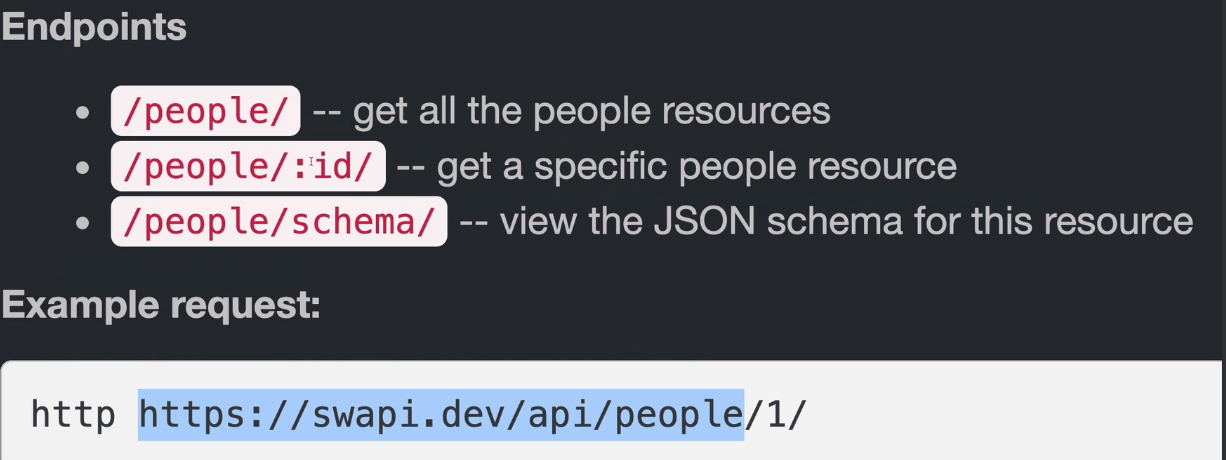
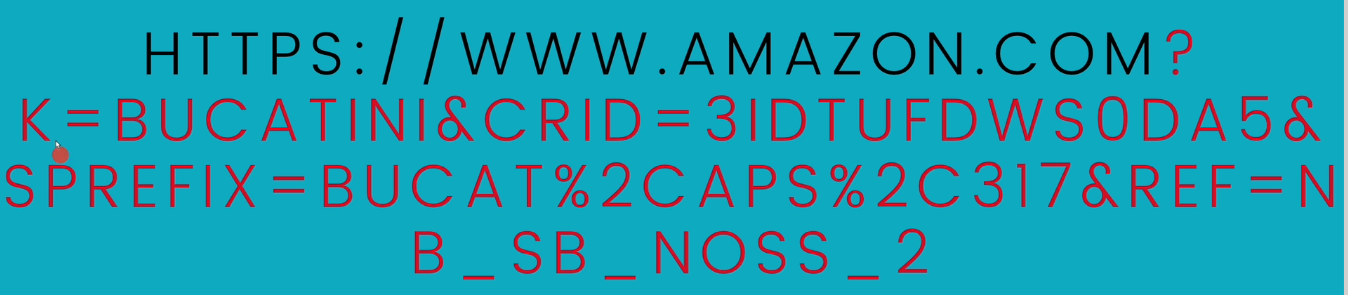
* **The Async Keyword**
  + 
  + The ‘**async**’ keyword makes a function into an ‘async function’
    - It will return a Promise Object (YOU don’t need to write the promise)
  + Function returns a value if the promise resolves
    - Else, returns the that promise was rejected
* **The Await Keyword**
  + 
    - Usually used in async functions
  + **Example**:
    - 
    - The await keyword makes the function ‘await’ the completion of the previous one first, before running
* **Use Try/Catch for handling errors in Async Functions:**
  + 

# Section 28: AJAX and APIs

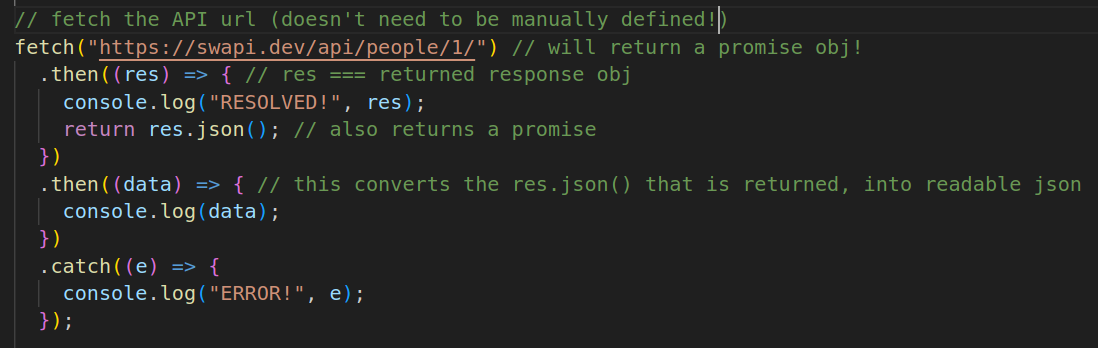
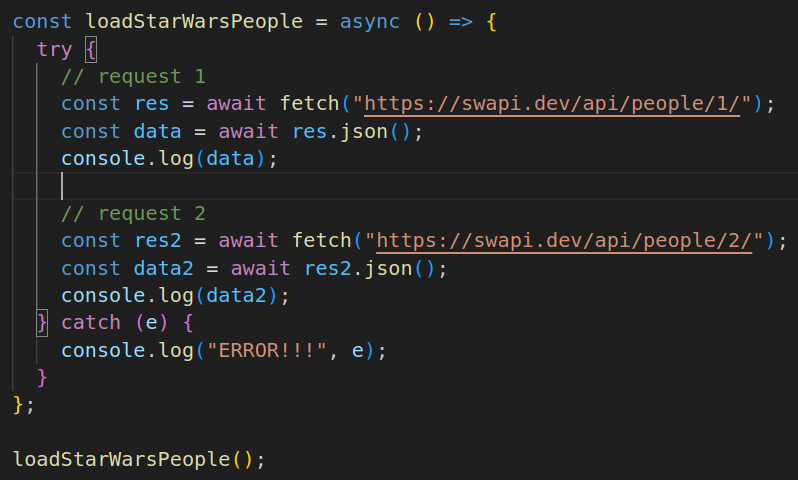
## Intro

* , but XML is no longer used, its JSON now
* JSON is consumed as a JS object
  + So when receiving the string for the JSON obj, you must first convert it into the object format (**JSON.parse(data)**)
  + 
* When converting JS object to string, use **JSON.stringify()**

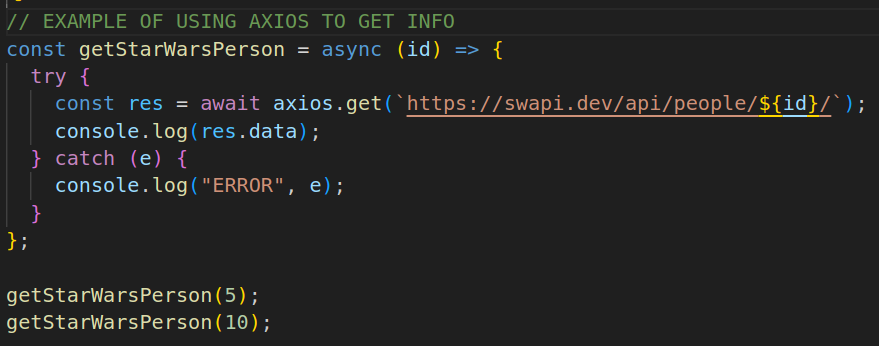
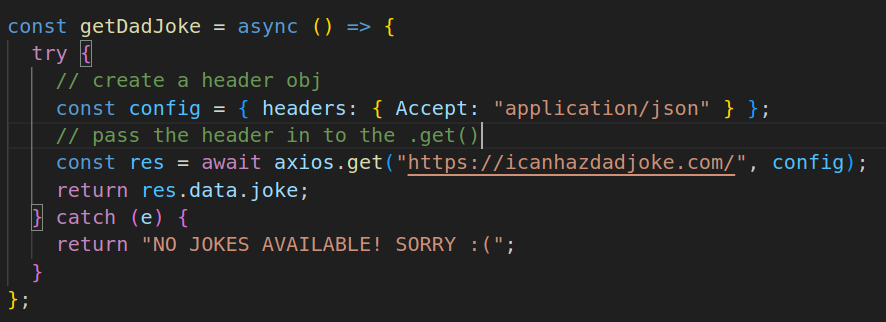
## HTTP Verbs/Status Codes

* Refer here for status codes: https://developer.mozilla.org/en-US/docs/Web/HTTP/Status
  + 200 is the BEST!
* **Understanding Query Strings**
  + API endpoints:
    - **Example**:
      * 
  + Combined with query strings to search for info!
    - **Complex Example**:
      * 
    - **Simpler example**:
      * 
    - HTTP Headers are key-value pairs that go with the request
      * Example:
        + 
        + **config** variable *reps the header object* that must be passed along with the request.
        + The purpose **depends** on the API, but can be used to identify what type of data you want back!

## Fetch API

* This API let’s us make requests via JS (supports promises)
  + This is already built-in so you don’t have to DEFINE it be used!
    - 
  + **Using fetch()**
    - 

## AXIOS (library for making HTTP reqs)

* Refer: [https://github.com/axios/axios#installing](https://github.com/axios/axios" \l "installing)
  + You can either ‘npm install’ the axios node module or use a CDN and insert into your HTML
    - <script src="<https://cdn.jsdelivr.net/npm/axios@1.6.7/dist/axios.min.js>"></script>
* **Example of axios**: *get a character’s info from api*
  + 
    - *axios.get()* -> returns a promise with the JSON parsed for you!
* You can also **pass** **headers** in with your .**get**()
  + 

# WHERE ARE PROJECTS LOCATED?

* Section 24 – Pokemon Project
* Section 26 – Score Keeper Project
* Section 28 – TV Show Project