http://bit.ly/workshop-attendance-19

http://bit.ly/WeatherApp19

Build a Weather App

DAY 4

Review Last Class

Let’s review Basic JavaScript!

Creating a JavaScript File

- JavaScript are linked to the HTML document using the <script> tag, just like how we linked the CSS to the HTML with the <style> tag!
  - Something like this:
  - `<script src="path/to/my/script.js"></script>`
- We can also learn JavaScript in the console!!! - We will talk about this later when we work on the weather app.

Variables

- Variables in JavaScript are dynamically typed
- JavaScript variables are declared using the *let* keyword:
    - `let a = 10; //This is a number with the value 10`
    - `let b = "hello world"; //This is a string with the value “hello world”`
- *Numbers* are used to represent numeric data (think 1, 2, 3, 4)
- *Strings* are used to represents a collection of characters (think a word, or a sentence)
- *let* are block-scoped, and *var* is functional scoped
    - TLDR: use let

Booleans

- *Booleans* are conditional statement, and can be produced using relational operators:
    - == (Equals To). Returns TRUE if both side are equal and FALSE otherwise
        - (3 == 3 => true, 3 == 4 => false)
    - && (And). Returns TRUE if both side are TRUE, and FALSE otherwise
        - 3 == 3 && 4 == 4 => true because TRUE && TRUE is TRUE
        - 3 == 4 && 4 == 4 => false because TRUE && FALSE is FALSE
    - || (Or). Returns TRUE if one side of the equation is TRUE.
        - 3 == 3 || 4 == 3 => true because 3 == 3 is TRUE
        - 3 == 2 || 4 == 3 => false because both sides are FALSE
    - ! (Not). Reverse the sign of a boolean value
        - !(True) == False => true (not true is false, false is equals to false)
        - !(3 == 2) && 3 == 3 => true (not 3 == 2 is true, and 3 == 3 is true, true and true is true)

Arrays

- Array is a collection of element, declared using the [ ] bracket.
    - `let food = ["apple", "orange", "banana"]; // an array of Strings representing food name`
    - `let numbers = [1, 2, 3, 4, 5]; // an array of numbers from 1 to 5`
- Access an array using the bracket notation and 0 indexing
    - `food[0] == "apple"`
    - `numbers[1] == 2`
- Adding to an array using push
    - `food.push("lemon");`

JavaScript Objects

- A one-dimensional sequence of values that are all stored in a single variable
- Instead of using an integer index as key (like an array), an Object uses *String*
  - Imagine a dictionary
- Object uses Key-Value pairs. Key can be used to look up values using the dot notation (.). You don't use position to refers to a key-value pair like an array.
- Objects uses { } to represents itself.
  - `let values = {"hritik" : 1, "kevin" : 2, "nam" : 3};`
  - `values.hritik == 1; values.kevin == 2; values.nam == 3`

If Else

- In JavaScript, you use conditional statement (if-else) for control structure
-
```
if (condition) {
    // do something
} else {
    // do something
}
```
- condition can be any expression that evaluates to a boolean value (true/ false)

For loop

-
  ```
  // an example for loop. The `i` is not declared as an int. This loops over the array and log

  // out elements at the ith position.

  let array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  for (let i = 0; i < array.length; i++){

    console.log(array[i]);

  }
  ```

While loop

-
  ```
  // an example while loop. This also loops over the array and log out elements at the ith position.

  // array.length = 10;

  let array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  let i = 0;

  while (i < array.length){

    console.log(array[i]);

    i++;

  }
  ```

Functions

- Declared using the *function* keyword in order to abstract code

```
//A function named `double` that takes 1 arguments

//and returns the doubled value of that argument

function double(num) {

  //Function body: perform tasks in here

  let doubled = num * 2;

  // Return: what you want the function to output

  return doubled;

}
```

```
// Call the double() function with the values 10

// Assign the result to `twenty`

let twenty = double(10);

// console.log(twenty) logs the number 20
```

Anonymous function

- In JavaScript, functions ARE variables:

```
let double = function(num) {

      return num * 2;

};

// console.log(double(10)) logs the number 20
```

- These produce the same function

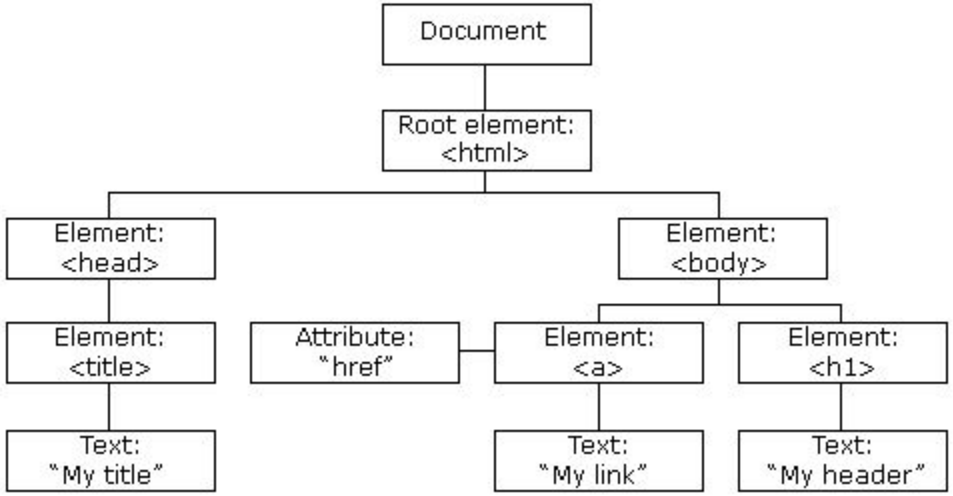
```
function foo(bar) {}

let foo = function(bar) {}
```

Document Object Model

Known as the “DOM”

Document Object Model (DOM)

- When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel.
- The HTML DOM Object is constructed as a tree of objects.
- The DOM basically gives us access to the HTML objects and allows us to manipulate it.



DOM Selectors

DOM selectors

- Allow us to access HTML element(s) in JavaScript using ids, classes, and other properties
- There are 5 types of selectors -
  - `getElementById()`
  - `getElementByClassName()`
  - `getElementByTagName()`
  - `querySelector()`
  - `querySelectorAll()`

getElementById()

- Used to access HTML elements by their ***Ids***
- Ex: *let* ***header_elem*** *=* ***document.getElementById("header");***

- A similar structure is used for `getElementByClassName()` and `getElementByTagName()`. But we won't discuss these in detail because we rarely use these.

querySelectorAll()

- Uses CSS selectors to access DOM elements.
- This will return ***all the DOM element*** matching the given selector ***in the form of an array*** of this HTML element objects.
- Ex:

  let elem = document.querySelectorAll("section p");

querySelector()

- Uses CSS selectors to access DOM elements.
- This will return ***only the first DOM element*** of this type. In simpler terms it returns the first element of the array that we get from `querySelectorAll()`
- Ex:

  let elem = document.querySelector("section p");

Manipulating DOM Elements

What can we manipulate?

- Change the HTML elements in the page
- Change the HTML attributes in the page
- Change the CSS styles in the page
- Remove existing HTML elements and attributes
- Add new HTML elements and attributes
- And much more…..

Change css properties using .style

We can change css styles of an element by using the ***.style*** property

```
let header_elem = document.getElementById(“header”);

header_elem.style.backgroundColor = “red” // change background color to red

header_elem.style.color = “red” // change text color color to red

header_elem.style.marginLeft = “20px” // change margin left distance

header_elem.style.display = “block” // set the display to block

header_elem.style.display = “none” // TRICK - you can hide elements with this
```

Change classes with .classList

- We can use the *.classList* property to change the class of an element

Ex: ***let header_elem = document.getElementById(“header”);***

header_elem.classList.add(“blue”); // add the class “blue” to #header_elem

header_elem.classList.remove(“blue”); // remove class “blue” from #header_elem

Remove elements

- We can use *.remove()* to remove an element from the DOM

Ex: ***let header_elem = document.getElementById("header");***

header_elem.remove(); // removes the first element with id as header_elem

Add elements

- We can use *.createElement()* to create a new element in the DOM

Ex:

```
let header_elem = document.getElementById(“header”);

// creates a new ‘paragraph’ element in document but still no position is given
let new_elem = document.createElement(“p”);

header_elem.append(new_elem); // adds our element to the end of header_elem
```

DOM Events

Call me if “this” happens

addEventListener()

- We can give specific behavior to our html elements like -

○ click	- When element is clicked
○ load	- What happens when the element loads
○ mouseover	- Same as hover in CSS
○	

Ex:

```
let header_elem = document.getElementById(“header”);

header_elem.addEventListener(“click”, my_fnc);
```

removeEventListener()

- We can remove the behavior we added before by simply using `removeEventListener()` the same way we used the `addEventListener()`

Ex:

```
let header_elem = document.getElementById(“header”);

header_elem.removeEventListener(“click”, my_fnc);
```

Any questions up till now?

Don’t feel shy!

Open exercise-4 folder into VS Code and read the prompt in the index.js file. No need to change .html

Switch to Kevin

APIs, Fetch, and JSON

What are APIs?

- APIs (or Application Programming Interfaces) allow us to access data from the web using HTTP requests

HTTP Requests

- We use HTTP requests to get data from the web
- Example: www.google.com/search?q=chipotle
- That request will render a website
- There are many types of HTTP requests, but we will focus mainly on GET requests

GET requests

- GET requests return data from APIs
- Example: https://samples.openweathermap.org/data/2.5/weather?q=London,uk
- https://samples.openweathermap.org is the root of the site

API Endpoints

- API endpoints are the base of a request
- Ex: https://samples.openweathermap.org/data/2.5/weather?
- This is the starting point for all requests to this API

Queries and parameters

- An API query is what data we want
- Ex: q=London,uk
- In a GET request, we pass this information in the URL
- q is the parameter and London, uk is the value we're passing it

Putting it all together

- When we go to this link, https://samples.openweathermap.org/data/2.5/weather?q=London,uk We'll get the current weather data for London
- Try opening this link in a new tab

What Happened?

You should've gotten back a result that looked like this:

```
{

    "cod": 401,
    "message": "Invalid API key. Please see http://openweathermap.org/faq#error401 for more info."
}
```

JSON Data

- The information returned from the request was JSON
- JSON stands for JavaScript Object Notation
- JSON is a standardized format for organizing data on the web
- Notice how the first 3 letters in JSON stand for “JavaScript Object,” JSON has the exact same syntax as JavaScript Objects!
- This makes it easy to use JSON data in JavaScript

Why didn’t the query work?

- A lot of APIs won’t allow you access to their data unless you have an API key
- This is a piece of information you send with your request to validate yourself
- To make a query with more than 1 parameter use the ‘&’ symbol
- Try adding this to the end of your earlier request:
  &appid=**7b9ef6d5a3c36d00b45d1c53aa1413c9**

Success!

```
{

    "coord": {
        "lon": -0.13,
        "lat": 51.51
    },
    "weather": [
        {
            "id": 300,
            "main": "Drizzle",
            "description": "light intensity drizzle",
            "icon": "09d"
        }
    ],
      ………………. . . . . . .
```

Note

- You can get your own API key for free from https://openweathermap.org/ by signing up
- For this workshop you can either of these keys:
- **7b9ef6d5a3c36d00b45d1c53aa1413c9**
- **3d35c1f672f998de9b32f06913d9d6c0**

Fetch

- So far we've only been sending requests through our browser
- We can send requests through JavaScript using the fetch function
- Example fetch call:

```
  const url = // some url you want to get data from
fetch(url).then(function(data) {
    // use .json() to parse the data
    return data.json()
})
.then(function(data) {
    // do something with the JSON
})
```

Let’s get back to our app now!

Follow Along!

That’s a lot for a day!

I think we should rest up a bit!