Tutorial Week 8

Introduction to Javascript

Javascript is the programming language of the web. With respect to the other 3 web-development languages:

1. **HTML** defines the **content** of a web page
    - Defines the information that we see on the page
2. **CSS** defines the **style** of a web page
    - Defines what certain things are supposed to look like
3. **Javascript** defines the **functionality** of a web page
    - Defines what happens when we click on things or input data

One common misconception is that Javascript is similar to Java. The reality is that the two are not connected in any sort of way other than the fact that both are programming languages. **Javascript is similar to Java in the same way a “car” is similar to a “carpet”**

In the past, Javascript used to be a very poorly written and exclusively frontend based language. With the improvements to the language, in the modern day, Javascript is very fast and is even used in some popular serverside technologies such as NodeJS

Javascript is coded similarly to other languages you might have seen (e.g. Python or Java) but it also has some “quirks” that make it unique (and *frustrating*) to work with at times

---

Including Javascript in HTML

Javascript can be included in HTML code using the `<script></script>` tag. Similarly to CSS, it can be included using either **inline Javascript** or using **external Javascript**

Inline Javascript:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
  <p>This is some content</p>
  <script>
```

```
    // we can write our javascript code here as we wish
    console.log('keviniscool');
  </script>
</body>
</html>
```
External Javascript:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
  <p>This is some content</p>
  <script src="script.js"></script>
</body>
</html>
```
There are two main differences with including Javascript when compared to including CSS:

1. We include both inline and external Javascript using the `<script></script>` tag
2. Javascript is included at the **BOTTOM** of the file as opposed to the `<head>` tag

The second point is very important difference between the two. Unlike CSS, Javascript **needs** to be added to the page after the HTML and CSS have loaded. If you do not do this, depending on the speed of your browser, the functionality of the site may load before you can actually see the page's components and you may get some unexpected results (e.g. you try to change things to elements that haven't loaded yet). CSS on the other hand, does not face this problem because even if it loads in first, it just waits for the HTML to finish loading to apply the styles. There is no consequence in the CSS loading before elements have fully loaded

---

Javascript Basics

NOTE: Javascript is actually a very complicated topic. In this tutorial (and in the course), we will only go over some very basic uses of Javascript. It is up to you to research and experiment with Javascript on your own. Javascript can do a lot more crazy things than just do things on click or modify simple things in HTML

Variables

Variables are defined using one of three keywords and the `=` operator:

1. `var x = 420`

2. `let x = 420`
3. `const x = 420`

`var` and `let` have minor differences in terms of scoping however for the sake of this course, their behaviour is essentially the same. It is recommended to use `let` over `var` nowadays however in the past (and in many older tutorials) you will see more uses of the keyword `var`

`const` defines a **constant** variable that will never change after being instantiated. You cannot change a `const`'s value after it has been instantiated otherwise Javascript will throw a TypeError

Similarly to Python, Javascript is dynamically typed meaning that you do not have to indicate the type of variable when you create it. Unlike Python, you have to prefix any variable you create with either `var`, `let`, or `const` to indicate that it is a variable

Equality

Equality is a tricky concept in Javascript. There are two main ways of checking for equality in Javascript using either the `==` or the `===` operators. Because of how weird Javascript typing is, the `==` operator will sometimes produce odd results. **It is recommended that you only use the `===` operator. Never use the `==` operator**

Some interesting cases of where `==` does something odd:

```
'' == '0'           // false
0 == ''             // true
0 == '0'            // true

false == 'false'    // false
false == '0'        // true

false == undefined  // false
false == null       // false
null == undefined   // true

' \t\r\n ' == 0     // true
```

Logical Operations

Javascript supports logical constructs similarly to other programming languages

`if`-`else` statement:

```
if (true) {
  console.log('keviniscool');
} else {
  console.log('this will never run');
}
```

`while` loop:

```
while (true) {
  console.log('keviniscool');
}
```

`for` loop:

```
for (let i = 0; i < 10; i++) {
  console.log('keviniscool');
}
```

Functions

Functions in Javascript are similar to most other programming languages. You declare a function by prefixing it with the `function` keyword:

Example:

```
function myFunction() {
  while (true) {
    return 'keviniscool';
  }
}
```

Arrays and Objects

An **array** in Javascript is similar to an array in Python. Much like in Python, Javascript arrays do not have to contain elements of the same type:

```
myArray = [1, "kevin", 2, true, {"firstName": "kevin", "lastName": "zhang"}];
```

An **object** in Javascript is similar to a dictionary in Python. Unlike in Python, the **keys** of the object must be **String** but the value can be any valid type. This type of structure is called a **JSON** or **Javascript Object Notation**

```
myObject = {
  "firstName": "kevin",
  "lastName": "zhang",
  "age": 69,
  "cool": true
}
```

Output and Modifying HTML

One of the most useful aspects of Javascript is being able to modify the content of an HTML file. Javascript can output to one of four areas:

- Writing to a pre-existing HTML element (using `innerHTML`)

- Writing to a new HTML element (using `document.write()`)
- Writing to an alert box (using `window.alert()`)
- Writing to the console (using `console.log()`)
    - You can view console output in Chrome by pressing `CMD+OPTION+J` on a Mac or `CTRL+SHIFT+J` on Windows

The most common way of obtaining an element to modify is by using `document.getElementById()`. Since IDs in HTML are guaranteed to be unique, we can access a pre-existing element in an HTML form using this strategy.

- There is a similar command called `document.getElementsByClassName()` however since classes are not unique, this command will always return the result in an array

Using this strategy, we can edit the content in an HTML file as follows

HTML (`index.html`):

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
  <!-- we will not see the text here because it will be overwritten by the
Javascript -->
  <p id="my-content">This is the original content</p>
  <script src="script.js"></script>
</body>
</html>
```

Javascript (`script.js`):

```javascript
document.getElementById('my-content').innerHTML = 'keviniscool';
```

Alternatively, we can create content that has never existed in the HTML before by using `document.write()` as follows

HTML (`index.html`):

```html
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Document</title>
</head>
<body>
  <!-- notice how there is no content in the body -->
  <script src="script.js"></script>
</body>
```

```
</html>
```

Javascript (`script.js`):

```
// after this happens, this HTML code will be added to the HTML file itself
// you can actually see this tag in the body by viewing the source in the
Chrome devtools
document.write('<p>keviniscool</p>');
```

Event Handling

In the past labs, we've made forms that don't do anything. Using Javascript, we can retrieve input from forms and perform interactive behaviour with the input we've collected. The most common way of doing this is by accessing the `.value` attribute of an element and reacting to an event such as `onclick()`

HTML (`index.html`):

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>My Website</title>
</head>
<body>
    <label>What is your name?</label>
    <input type="text" id="name-form" placeholder="Enter your name">
    <button type="submit" onclick="nameResult()">Submit</button>
    <!-- we put a placeholder <p> here so that it can be written into later -
->
    <!-- using document.write() would overwrite the entire body -->
    <p id="result"></p>
    <script src="script.js"></script>
</body>
</html>
```

Javascript (`script.js`):

```
// this function is called whenever the button is clicked
function nameResult() {
  // get the value inside the form
  name = document.getElementById('name-form').value;
  // add the value
  document.getElementById('result').innerHTML = 'your name is: ' + name;
}
```

Frontend vs Backend

You may be wondering (if you've been reading the handout) what the difference between using Javascript is over Flask. The tutorial we did a few weeks ago on Flask could have been completely accomplished using Javascript alone so why do we even use Flask?

There are several things that we cannot accomplish if we use Javascript alone:

- If we want to send data back to a server or retrieve data from a database
- Javascript is only meant to handle client interactions, it is not supposed to logically process data
    - Example: If you were making a game, the Javascript is only there to process that a character wants to move. It does not process whether or not it was legal or not for that character to move
- Javascript is intentionally limited for security reasons
    - Example: Javascript cannot read or write files (Flask can)

Exercise

The starter code can be found here

Using the starter code found above, we will be making an implementation of the game Rock-Paper-Scissors. The game will be singleplayer and you will play against the computer who makes random moves. The game works as follows:

- The player clicks one of the three options (rock, paper, or scissors)
- Three messages are created:
    0. A message indicating the player's move (e.g. **The PLAYER chooses: ROCK!**)
    1. A message indicating the computer's move (e.g. **The COMPUTER chooses: SCISSORS!**)
    2. A message indicating the outcome of the game (e.g. **The PLAYER wins!**)
        - It is possible for the game to end in a tie if both players choose the same option
- Clicking again will play through a new game

A couple notes to help you get started:

- There is a function called `getRandomInt(max)` that generates a random number between 0 and `max` exclusive (e.g. `getRandomInt(3)` would generate a random number between 0 and 2) that can help you generate a random move for the computer to play

- The HTML and CSS code is completely written for you so no changes will be required
    - Look at how the `play()` function is implemented to help you complete the code
- We want to append `<li>` elements to the `<ul>` element with the ID `game-content` in the HTML. We can do this in Javascript as follows:

Appending to a pre-existing `<ul>` element:

```
// get the actual element
let gameContent = document.getElementById('game-content');

// create a list item
let listItem = document.createElement('li');
// add some content to this list item
listItem.appendChild(document.createTextNode('this is some text'));

// add the list item to the element
gameContent.appendChild(listItem);
```

This will create the following HTML code:

```
<ul id="game-content">
  <!-- in the beginning, this ul element will be empty -->

  <!-- after running the Javascript code above, we will get this line -->
  <li>this is some text</li>
<ul>
```

References

- Javascript documentation