CMPT 361: JavaScript (JS) Tutorial

We will go through the contents of this tutorial in the class.

What is Javascript? (source: link)

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved.

- HTML is the markup language that we use to structure and give meaning to our web content, for example defining paragraphs, headings, and data tables, or embedding images and videos in the page.
- CSS is a language of style rules that we use to apply styling to our HTML content, for example setting background colors and fonts, and laying out our content in multiple columns.
- JavaScript is a scripting language that enables you to create dynamically updating content, control multimedia, animate images, and pretty much everything else. (Okay, not everything, but it is amazing what you can achieve with a few lines of JavaScript code.)

Let's take a simple text label as an example. We can mark it up using HTML to give it structure and purpose:

```
<p>Player 1: Chris</p>
```

Player 1: Chris

Then we can add some CSS into the mix to get it looking nice:

```
p {
  font-family: "helvetica neue", helvetica, sans-serif;
  letter-spacing: 1px;
  text-transform: uppercase;
  text-align: center;
  border: 2px solid rgb(0 0 200 / 0.6);
  background: rgb(0 0 200 / 0.6);
  color: rgb(255 255 255 / 1);
  box-shadow: 1px 1px 2px rgb(0 0 200 / 0.4);
  border-radius: 10px;
  padding: 3px 10px;
  display: inline-block;
  cursor: pointer;
}
```

And finally, we can add some JavaScript to implement dynamic behavior:

```
const para = document.querySelector("p");

para.addEventListener("click", updateName);

function updateName() {
 const name = prompt("Enter a new name");
 para.textContent = `Player 1: ${name}`;
}
```

Run this and try clicking on the last version of the text label to see what happens (you can also find this demo on GitHub — see the source code, or run it live)!

Software Recommendations

1. A code editor such as Visual Studio Code.
2. A local http server for viewing your web application during development. One such option is to use the Live Preview extension for VS Code. Another option is to create your own http server using Node.js or Python. A Python file to do this has been provided for you.
3. A browser for debugging such as Edge, Chrome, or Firefox.
4. Version control such as Git, along with Git LFS for storing large binary data files. The GitLens extension is also helpful if working in VS Code.

General Information

1. In general it is a good idea to test your final submission to ensure it runs as expected and that all required files are present.
   1.1. Make sure to include any setup steps if necessary so that we can run your code.
2. Sometimes you may need to clear the cache when refreshing to view your changes in a browser. This can usually be done with **CTRL + F5**.
3. Please use version control such as Git and commit frequently so you can revert to a working version of your project as and when needed.

Project Setup

1. Create a new folder for your project and extract the files from the course website.
2. Initialise a new Git repository:

```
git init
git config user.name 'Your Name'
git config user.email your_email_id@sfu.ca
```

3. Set up Git LFS for your repository.
```
git lfs install
git lfs track *.png *.jpg *.pdf
```
4. Add a .gitignore file to the root of the repository:
   4.1. Each line specifies a pattern for files and folders to ignore. Patterns can be specified using a wildcard (`*`).
   4.2. Add **`.vscode`** on the first line (or a different file/folder if using another editor).

Modules

1. JavaScript programs can be split into modules across multiple files.
2. The `import` keyword is used to access code from other modules.
3. The `export` keyword makes functions and variables externally accessible.
4. This functionality requires an **HTTP** server.

JavaScript Basics

1. Creating a web page and embedding a JavaScript file:
   1.1. The main page of your web application should be an **`index.html`** file.
   1.2. The **`<script>`** tag can be used to embed a JavaScript file into your web page. You may also want to tell the browser to treat the file as a module to import and export code.
```
<script src="tutorial.js" type="module"></script>
```
   1.3. A style sheet can also be included:
```
<link rel="stylesheet" href="style.css">
```

2. We can enable strict mode for a JavaScript file by writing the following at the top:
```
"use strict";
```
If your script is loaded as a module, strict mode should be enabled by default.

3. Displaying messages:
```
console.log("Hello!");
alert("Goodbye!");
```

4. Declaring variables and constants:
```
var x = 5;  // scoped to function body
let y = 10;  // scoped to the block
const z = 15;  // scoped to the block
```

5. JavaScript Primitives:
   5.1. Boolean
   5.2. Number (double)
   5.3. BigInt (arbitrary precision)
   5.4. String
   5.5. Symbol (unique identifier)

5.6. Null
5.7. Undefined

6. JavaScript allows grouping of data into objects:

```
let car = {colour: "green", year: 2003};
```

7. Classes can be defined with the following syntax:

```
class Car {
   constructor(colour, year) {
       this.colour = colour;
       this.year = year;
   }
   method_1(...) {
       ...
   }
}
```

8. Classes can be instantiated using the **new** keyword:

```
let x = new Car("green", 2003);
```

9. Arrays can be created as indicated below and can contain elements of different types:

```
let array = ["cat", 50, false];
array[0];  // Accessing array elements
array.at(-1);  // Accessing the last element
array.push(element);  // Append elements to the array
array.unshift(element);  // Elements can also be prepended
array.pop();  // Remove and return the last element
array.shift();  // Remove elements from the front
array.length;  // The number of elements in the array
let a2 = array.map(element => element + 1);  // Transform arrays
```

10. Example of a key-value Map:

```
let map = new Map();
map.set("some key", someValue);
let y = map.get("some key");  // y = someValue
map.has("some key");  // returns true
map.delete("some key");
map.clear();
map.size;  // returns 0
```

11. Control Flow:
    11.1. If-else

```
let grade = 85;
if (grade >= 90) {
```

```
 console.log("A");
} else if (grade >= 80) {
 console.log("B");
} else {
 console.log("F");
}
```

11.2. Strict equality operator (===) considers operands of different types to be unequal.

11.3. Switch statements

```
switch(expression) {
   case x:
     // code block
     break;
   case y:
     // code block
     break;
   default:
     // code block
}
```

11.4. for loop

```
for (let i = 0; i < 5; i++) {
   console.log(i);
}

const numbers = [45, 4, 9, 16, 25];
for (let i in numbers) {
   console.log(i);
}
```

11.4.1. Skip to the next iteration using the `continue` keyword.

11.4.2. The loop can be interrupted using the `break` keyword.

11.5. while loop

```
let i = 0;
while (i < 5) {
   console.log(i);
   i++;
}
```

11.6. do-while

```
let i = 1;
do {
   console.log(i);
   i++;
} while (i <= 5);
```

12. Functions can be defined with the **`function`** keyword. For example:

```
// Function to compute the product of p1 and p2
function myFunction(p1, p2) {
   return p1 * p2;
}
```

12.1. Anonymous functions (without a name), can be passed as parameters to other functions.

13. Modifying HTML with JavaScript:

```
document.getElementById("myElement").innerText = "Welcome!";
```

14. Modifying CSS:

```
document.body.style.backgroundColor = 'SlateGray';
```

15. Hiding elements:

```
document.getElementById("myElement").style.display == "none";
```

16. Showing elements:

```
document.getElementById("myElement").style.display == "block";
```

17. Buttons can be described by the HTML and respond to events:

```
<button type="button" onclick=myFunc()>My Button</button>
```

18. We can also specify event handlers in JavaScript:

```
document.getElementById("myElement").onchange = function(event) {
   ...
}
```

19. Errors can be handled using try-catch-finally:

```
/*
"alert" is misspelt as "adddlert" deliberately to produce an
error.
*/
try {
   adddlert("Welcome guest!");
}
catch(err) {
   document.getElementById("demo").innerHTML = err.message;
}
finally {
      ...
}
```

19.1. Code in the try block may generate an error, which can be handled in the catch block. The finally block runs after try-catch.

20. Errors can be thrown using the following syntax:

```
throw "Too big";    // throw a text
throw 500;          // throw a number
```

21. Timeouts can be used to call a function after the specified number of milliseconds:

```
id = setTimeout(handler [, timeout [, ...arguments ] ]);
clearTimeout(id);
```

22. Use intervals to call a function every time the specified number of milliseconds elapses:

```
id = setInterval(handler [, timeout [, ...arguments ] ]);
clearInterval(id);
```

Exercises

The following exercises can be completed in the provided tutorial.js file.

1. Logging messages.
2. Creating objects and classes.
3. Working with arrays and maps (in the console).
4. Creating a button and responding to an event.
5. Reading text input.
6. Responding to events from menus and radio button forms.
7. Reading the value of a slider when it changes.
8. Changing the background colour using a colour picker.
9. Processing keyboard and mouse input events.
10. Calling functions at intervals.

Debugging

1. It is possible to attach a debugger to your JavaScript code in order to interact with breakpoints and view error output.
2. Debugging with only a browser:
    a. Open the page in a browser of your choice.
    b. Open the Inspector / DevTools **(CTRL + SHIFT + I)**.
    c. From the Debugger tab (Firefox), you can add and step through breakpoints in your JavaScript code. In Chrome, this is the Sources tab.
    d. The logged output and error messages can be viewed in the Console.
3. You can also start debugging from VS Code:
    a. Install both the Live Preview and JavaScript debugger extensions.
    b. Start the live preview server by opening the command palette **(CTRL + SHIFT + P)** and selecting “Live Preview: Start Server”.
    c. Open the command palette and select “Debug: Open Link” and enter the address of the live preview server. By default this is http://localhost:3000.

Additional Resources

1. The Modern JavaScript Tutorial
2. W3 Schools JavaScript Tutorial
3. MDN Web Docs
4. Chapter 3, Interactive Computer Graphics: A Top-Down Approach with WebGL