FIT3179 DATA VISUALISATION

Week 7 Studio Activity (Part A): Introduction to JavaScript

1. Introduction to JavaScript

JavaScript is a Client-side scripting language. JavaScript code is downloaded to the user's computer to be executed in the browser. This has advantages:

- The browser can process user interaction events in real-time.
- Code can execute without using server resources.

1.1 Syntax of JavaScript

JavaScript statements are placed within the **<script>... </script>** HTML tags in a web page. JavaScript uses a C/Java-like syntax. Each individual statement in JavaScript should be separated using semicolons. Blocks of code should be written between a pair of {}. A block of code is used for declaring functions, loops, etc.

```
var end = 10;                       // an individual statement ()
for (let i = 0; i < end; i++) {     // start of a block of code with one statement
   console.log(i * 2)               // a single statement inside the block of code
}                                   // the end of the the block of code
```

Figure 1. Basic JavaScript code

1.2 Saving and Embedding JavaScript files

There are two methods to embed a JavaScript file in HTML code:

- Internal Embedding: JavaScript can be embedded directly into the HTML document by placing the JavaScript code into a <script> element, which is best placed in the <head> element or at the end of the document just before closing the <body> element as shown in the first <script> element in Figure 2.
- External Embedding: JavaScript can be written in a separate document and can be embedded into the HTML document by using the <script> element and assigning the link to the .js file to the **src** attribute as shown by the second <script> element in Figure 2.

```
<head>
    <title>JavaScript in HTML</title>

    <!-- Internal Embedding -->
    <script>
        var number = 10;

        alert("The number is: "+number);
    </script>

    <!-- External Embedding -->
    <script src="script.js" ></script>
</head>
```

Figure 2. Internal and external embedding

2. Adding Interactivity with JavaScript

2.1 Variables in JavaScript

Variables in JavaScript can be declared with the **var** keyword as shown in the example in Figure 3. Variables are containers for storing data (values). The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits, underscores, and dollar signs.
- Names must begin with a letter
- Names are case sensitive (Abc, ABC and abc are different variables)

It is recommended that you use the camel case convention when naming variables, for example, `getElementById` instead of something like `get_elementBy_ID`.

The value of any variable can be printed to a console with console.log(variableName), see Figure 1. Every browser has developer tools that are accessible through your browser's menu; one of these tools is the console.

```
var message; // variable declaration
message = "FIT3179 - Data Visualization"; // assigned a string value
alert(message); // access a variable
```

Figure 3. A simple example of declaring and using variables

2.2 Triggering Events in JavaScript

A JavaScript function is defined with the **function** keyword, followed by the name of the function which is then followed by parentheses () as shown in Figure 7. A JavaScript function is executed when some other code invokes it (calls it) or it is called by itself. There can be multiple or no arguments at all in the parentheses. The code that is placed in the {} is executed line by line in the order it has been written.

In this example, we will demonstrate a JavaScript function that is called when a button is pressed. The function will increment a counter value each time it is called and display the number of button clicks in a element. This example will also demonstrate how to toggle the visibility of an element when a text element is clicked.

Step 1: As shown in Figure 4,

- Create a new HTML document and save it as events.html.
- Under the <**head**> element, add the events.css file
- Inside the <**body**> element, create a <**h1**> element, a <**button**> element and a <**span**> element to demonstrate click events getting triggered on the click of the button. Add a *numberOfClicks* attribute to the <**span**> element and set its value to 0. This custom *numberOfClicks* attribute will store the number of times the button has been clicked and will be updated every time the JavaScript function shown in Figure 8 is called.
- Just before closing the <**body**> element, add a <**script**> element and in the src attribute mention the name of our script file (events.js).

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Triggering Events in HTML</title>

   <!-- The style sheet applied to this document. -->
   <link rel="stylesheet" href="events.css">
</head>
<body>
   <h1>Triggering click events on HTML elements</h1>

   <!-- A click on this button will trigger an event. -->
   <button id="incrementButton">Increase the Counter</button>
   <!-- The text and the numberOfClicks attribute change when the button is clicked. -->
   <span id="counter" numberOfClicks="0">0</span>

   <!-- This .js file is loaded and then runs after all other elements are loaded. -->
```

```
    <script src="events.js"></script>
</body>
</html>
```

Figure 4. Structure of the events.html document

Step 2: Create a new CSS file and save it as events.css. Include the following code to set an appropriate spacing between the <**button**> and <**span**> element.

```
#counter {
    margin-left: 30px;
}
```

Figure 5. Structure of events.css file

Step 3: Now let's add the JavaScript code to handle events. Create a new JavaScript file and save it as events.js

- Create two JavaScript comments by adding // in front of a line as shown in Figure 6. By doing this, it is easy to separate function declarations and function calls.

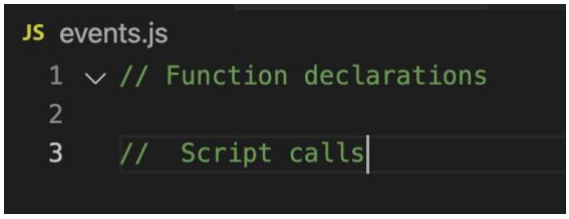

Figure 6. Comments in JavaScript

- Declare a function that will handle the click event. Call this function *incrementCounter*. This function first retrieves the span element, then reads the *numberOfClicks* attribute of the span element, increments that value by one, and finally updates the text and the *numberOfClicks* attribute of the span element.
    - To retrieve an element, we use the predefined JavaScript function **document.getElementById('yourID')**. We have set the id attribute of the element in the HTML document to "counter" (see Figure 4). So use document.getElementById('counter');
    - Use the predefined **Element.getAttribute('attributeName')** function to get the value of the *numberOfClicks* attribute and store it in a count variable: var count = element.getAttribute('numberOfClicks');
    - Increment the value of the count variable by one.
    - **Element.innerText** is the textual content displayed by a span element. Assigning a new value to the innerText attribute modifies the text, in this example, the new text is the *count* value.
    - Use **Element.setAttribute(name, value)** to replace the value of our custom *numberOfClicks* attribute. The initial value that we set when defining the *numberOfClicks* attribute was 0. Each time the incrementCounter function is run, the value of the *numberOfClicks* attribute will be increased by 1.
    - Figure 7 shows the entire function. Make sure you understand each step in this function.

```
function incrementCounter() {
    var spanElement = document.getElementById('counter');
    var count = spanElement.getAttribute('numberOfClicks');
    count++;
    spanElement.innerText = count;
    spanElement.setAttribute('numberOfClicks', count);
}
```

Figure 7. Function declaration for incrementing counter

- In the script initialisation section, we need to assign the *incrementCounter* function as an event handler to the button. Again use the getElementById function to retrieve the button. Then add the

incrementCounter function as a handler for click events to the button using the predefined target.**addEventListener('eventname', listener)** function. The first parameter for this function is the type of event we want to listen to, which is 'click' in our case. The second parameter is the function that will be called when the event occurs, which is our incrementCounter function (see Figure 8).

```
// Script calls
document.getElementById('incrementButton').addEventListener('click', incrementCounter);
```

Figure 8. Assigning the *incrementCounter* function as an event handler to the button

- Make sure the *incrementCounter* function is now called every time the button with ID 'incrementButton', and you see the text increment by 1 on each button click.
- **Toggling visibility of elements**: Let's add an image that is shown or hidden when a text element is clicked. You can later extend this example to swap two or more images (or other HTML elements).
- In the **events.html** file, create a new ** ** element as a container to hold an **<img>** element. Add the path of the **toggle.png** image file into the **src** attribute of the image element as shown in Figure 9. Also add a **<p>** element. When this text is clicked, the image will change visibility. Assign a unique id to the div, the img and the p elements.

```
<div class="chartContainer">
      <p id="chartToggle">Click to toggle chart visibility</p>
      <img src="images/toggle.png" id="chart1">
 </div>
```

Figure 9. Creating a new container to toggle the visibility of an image

- In the **events.css** file, add the following code to give the container an appropriate margin and restrict the size of the image (Figure 10).

```
.chartContainer {
   margin-left: 50px;
   margin-top: 100px;
}
img {
   width: 400px;
}
```

Figure 10. events.css file

- In the **events.**js file, create the function named **toggleVisibility()**, which will be the event handler. Eeach HTML element has a style attribute (which can be defined by CSS). One child attribute of this style is called display, which can be "none" or "block". A value of display:none means the element is not rendered and no space is reserved for it between the other elements. A value of display:block is the opposite, that is, space for the element is "blocked" and the element is rendered.

```
function toggleVisibility() {
   var chartImage = document.getElementById('chart1');
   if (chartImage.style.display === "none") {
       chartImage.style.display = "block";
   } else {
       chartImage.style.display = "none";
   }
}
```

Figure 11. Toggle Visibility function

- As shown in Figure 11, the variable **chartImage** is selecting the image with the ID attribute **chart1**. With **.style.display** the value of an element's display type is either set or read.

- The IF-ELSE conditional in Figure 11 compares the element's display property with "none" and "block" strings. The code after the IF check the style.display property to **"block"** (if the element is currently hidden), else style.display is set to **"none"**.