

FIT3179 DATA VISUALISATION

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

1. Introduction to JavaScript	2
1.1 Syntax of JavaScript	2
1.2 Saving and Embedding JavaScript files	2
2. Data Types in JavaScript	3
2.1 An Easy Practice Window	3
2.2 Primitive values	4
2.3 Objects	6
2.4 Adding numbers and adding strings	6
2.5 Convert string to number or number to string	7
2.6 Date data type	7
3. Adding Interactivity with JavaScript	9
3.1 Variables in JavaScript	9
3.2 Triggering Events in JavaScript	9

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 code in an `<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 in the HTML document by using the `<script>` element that links to the .js file using 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. Data Types in JavaScript

There are many different built-in data types in JavaScript. This section attempts to list most of the built-in data types available in JavaScript with their properties.

2.1 The Console Window for Practicing

One easy way to test JavaScript commands is to use the console window from the developer tools of your web browser. To open it in Chrome, press **F12** if you are using Windows, or you can press **option + command + I** if you are using macOS.

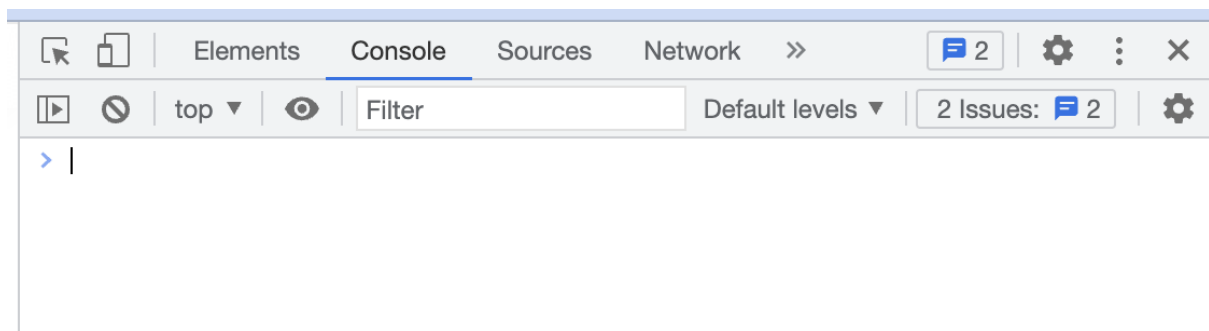


Figure 3. Console Window in Chrome Browser.

JavaScript has [primitive value](#) types and [object](#) types. You can use the `typeof` function to check the data type of a variable.

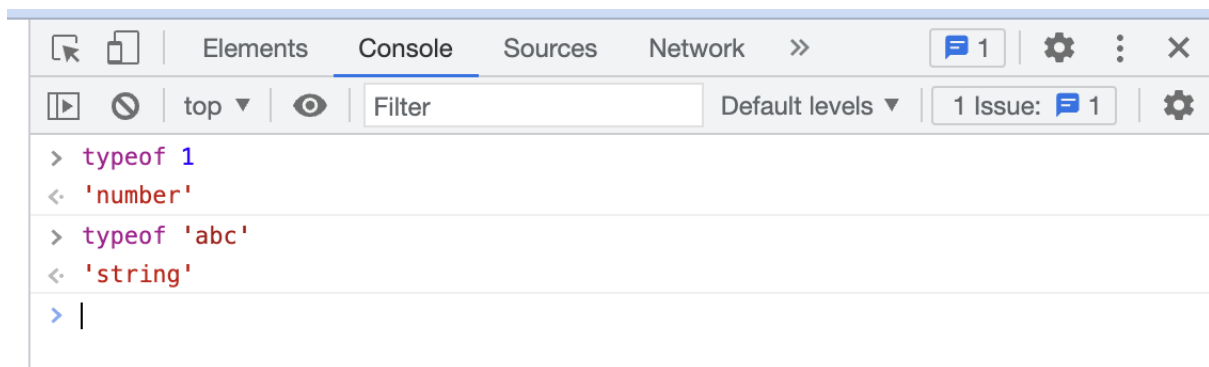


Figure 4. The `typeof` function returns the type of a variable.

2.2 Primitive values

- [Number type](#) (number and NaN)

JavaScript has only one number type; all numbers are always 64-bit floating point.

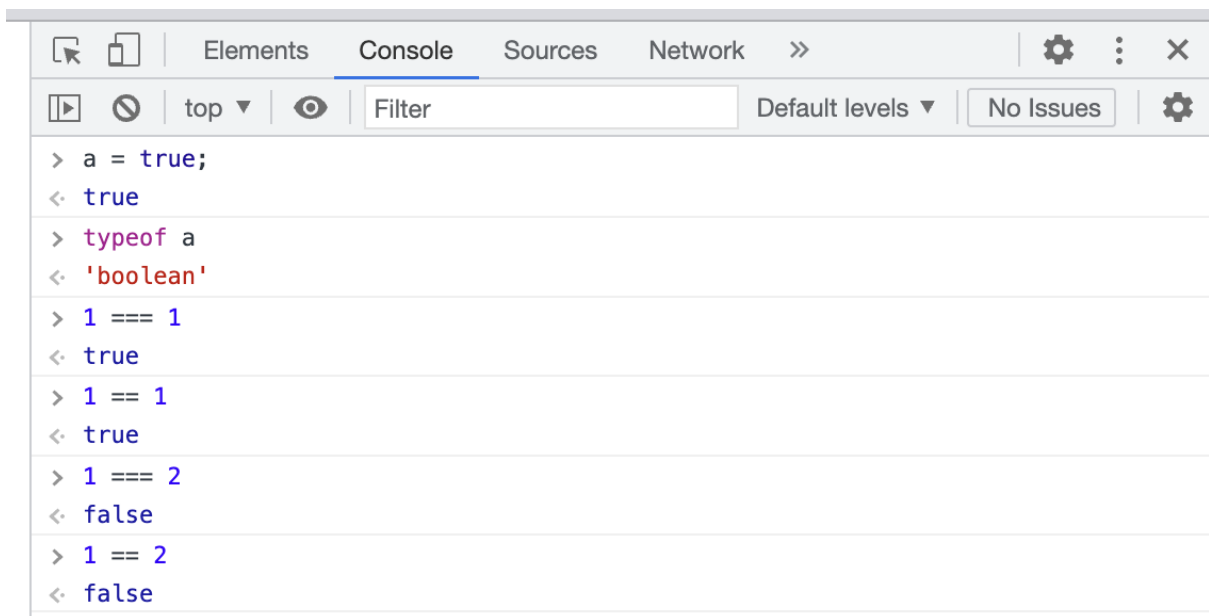
NaN is short for “Not a Number”. It is the result of an arithmetic operation that cannot be expressed as a number, for example, a division by zero. It is also the only value in JavaScript that is not equal to itself.

- [Boolean type](#) (true and false)

true and false can be used to determine the output of logical operations. For example, if your operation is 1 equal to 1, then the output will be true, if your operation is 1 equal to 2, then the output will be false. The == operator compares two variables and returns true if they have the same value. The === operator compares values across different data types.

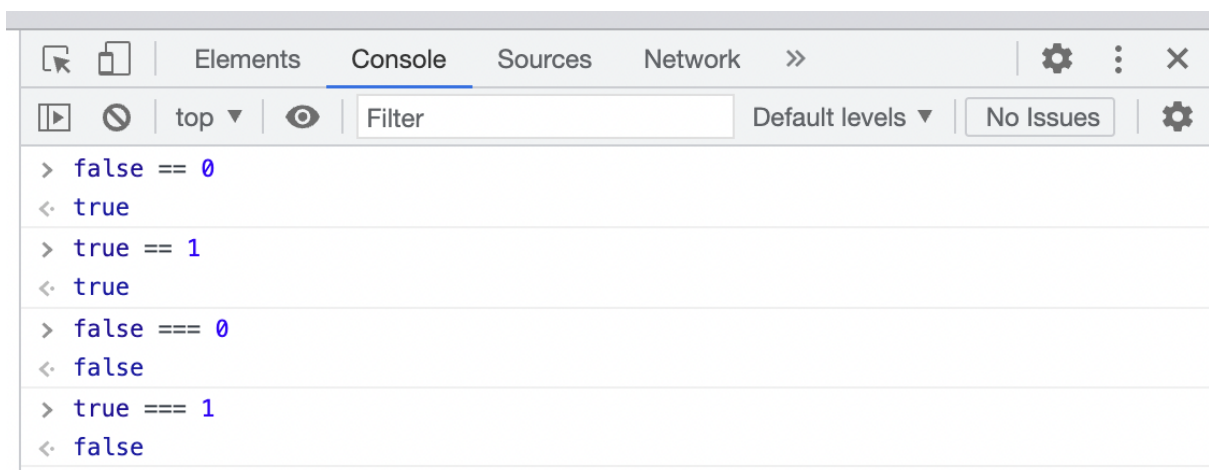
If you want to compare value and data type at the same time, use the === operator. Numerical numbers 1 and 0 can be interchangeable for true and false.

In the console, compare false with 0 using the == operator and the === operator. Make sure you understand the different results.



```
> a = true;
< true
> typeof a
< 'boolean'
> 1 === 1
< true
> 1 == 1
< true
> 1 === 2
< false
> 1 == 2
< false
```

Figure 5. Usage of == and === in JavaScript



```
> false == 0
< true
> true == 1
< true
> false === 0
< false
> true === 1
< false
```

Figure 6. **Equality** of boolean data and numeric data.

- [Null type](#) (null)

Null in general means nonexistence or invalid, and the data type of null is an object.

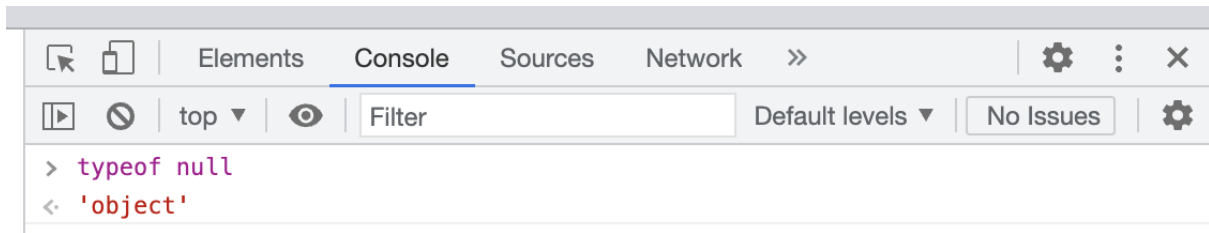


Figure 7. Null is defined as an object type in JavaScript.

- [Undefined type](#) (undefined)

Undefined means that data hasn't been assigned to anything. It commonly happens if you try to access data or objects with an invalid name.

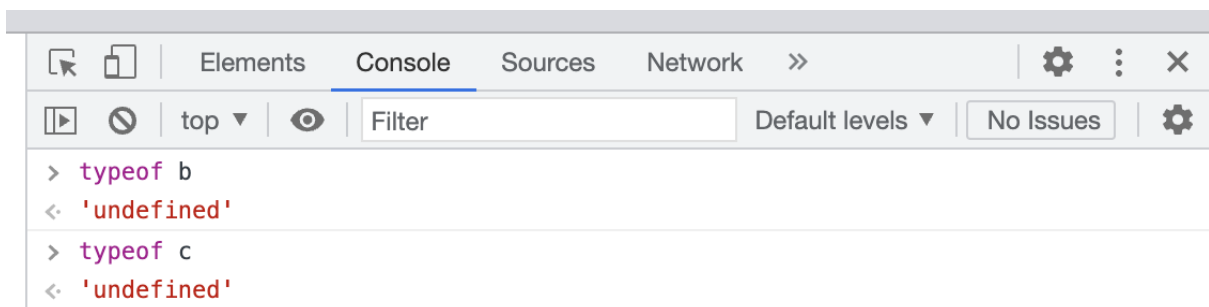


Figure 8. Variables without value assignment are undefined.

- [String type](#) (characters include numbers, alphabet, and special characters) JavaScript's string type is used to represent textual data.

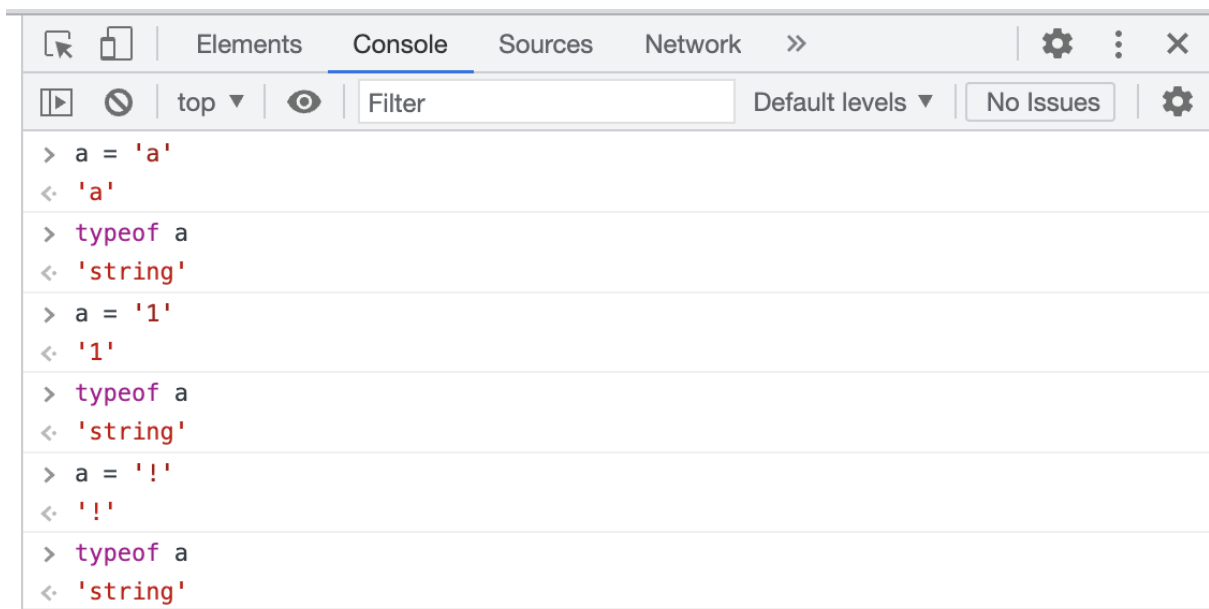


Figure 9. Different characters for string type data.

2.3 Objects

In JavaScript, objects can be seen as a collection of properties.

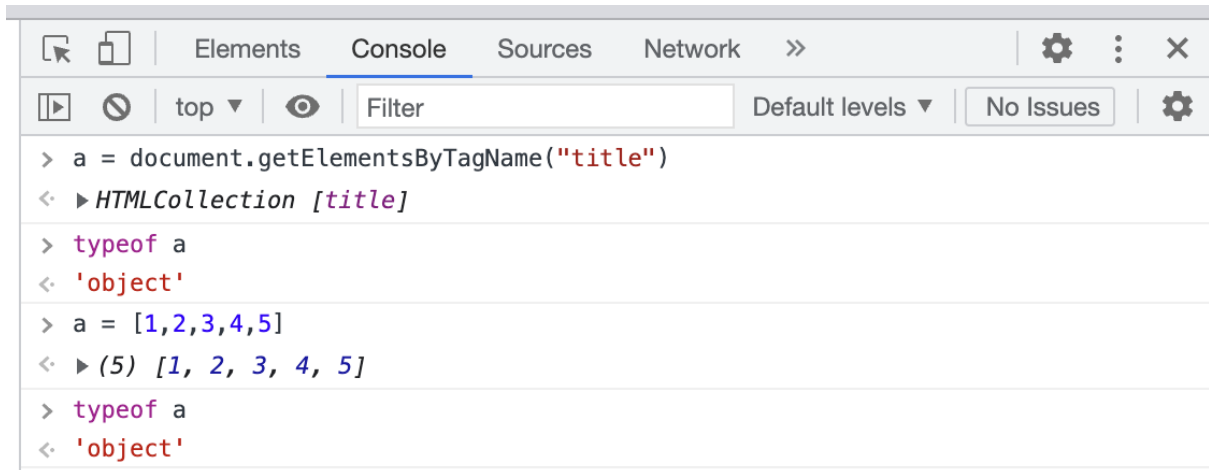


Figure 10. Examples of object type data.

2.4 Adding numbers and adding strings

Adding numbers will compute the arithmetic sum; while adding strings will concatenate the individual strings together and create a new string. Adding numbers and strings together might not lead to an error in JavaScript, but it is important to make sure the data type of all items is the same before you add them together.

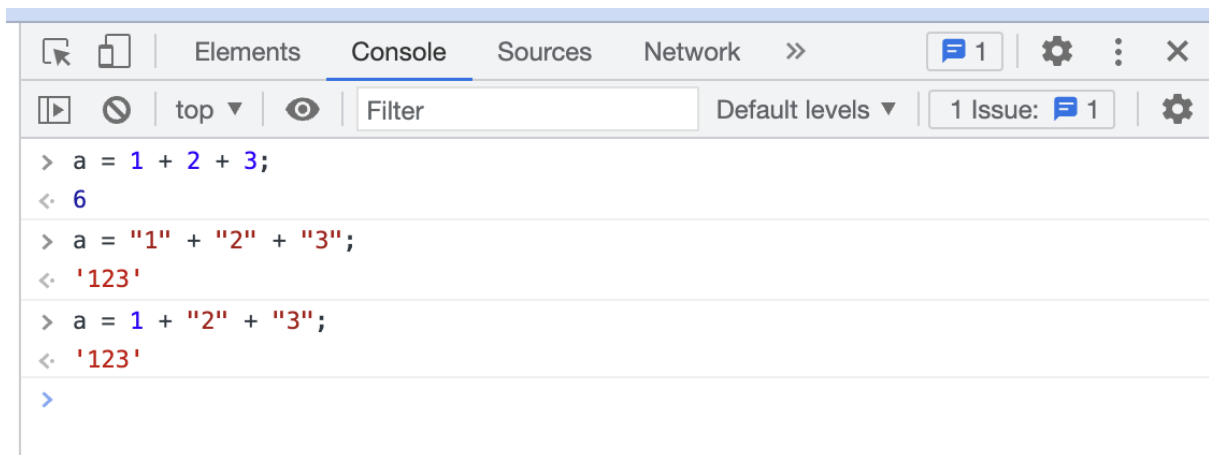
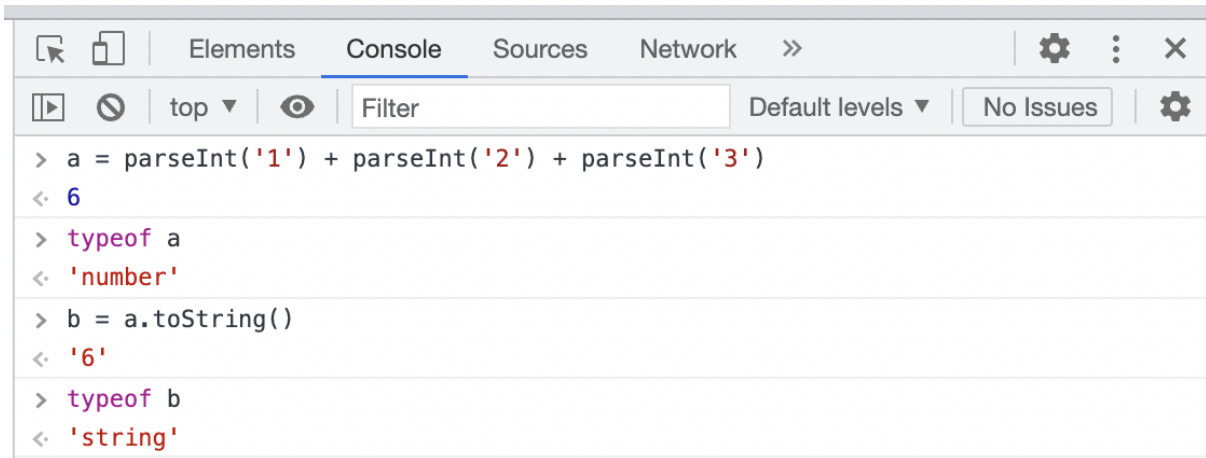


Figure 11. Example of adding numbers and adding strings.

2.5 Convert string to number or number to string

You can convert the data type from string to number by using `parseInt(string)` or `parseFloat(string)`. You can also convert a number into a string by using `YourNumber.toString()`.



```
> a = parseInt('1') + parseInt('2') + parseInt('3')
< 6
> typeof a
< 'number'
> b = a.toString()
< '6'
> typeof b
< 'string'
```

Figure 12. Convert number to string or string to number in JavaScript.

Exercise 1:

Create two variables `a` and `b`, and set `a = 1` and `b = 10`. Convert `a` and `b` to two string variables with new names using `toString`. Then use the `+` operator to create variables `c`, `d`, `e`, and `f` with the values and types as listed below. You may use `parseInt` to create `f`.

`c = 11`

`d = '11'`

`e = '101'`

`f = 101`

2.6 Date data type

The `parse()` method takes a date string (such as `"2011-10-10T14:48:00"`) and returns the number of milliseconds since January 1, 1970, 00:00:00 UTC. You can also create a new date with a specific year, month, and day.

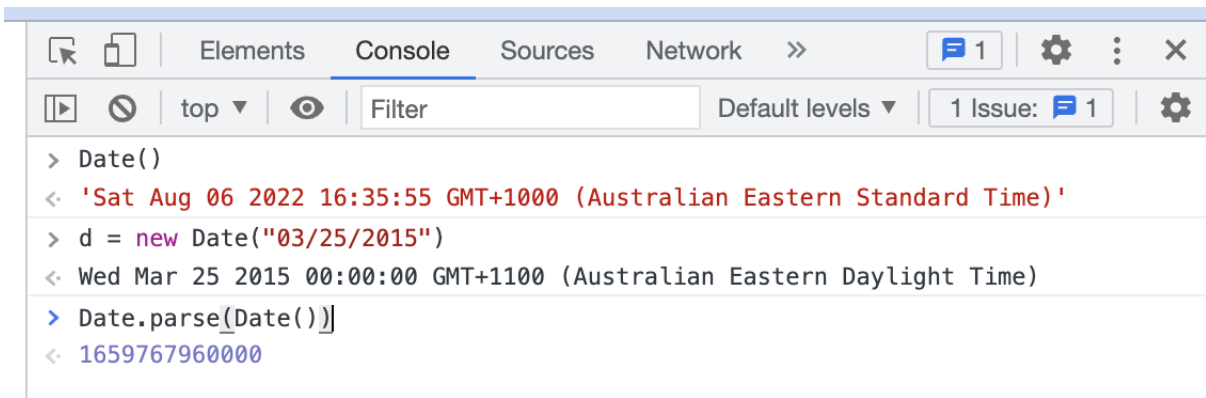


Figure 13. Create a date object or get the current date in JavaScript.

- Date format

The date object can be formatted into a specific style. For example, you can use `DateObject.toLocaleString()` to convert the date into the “day-month-year, time” format, or you can use `DateObject.toString()` to convert the date into the “weekday month day year” format.



Figure 14. Usage of `toLocaleString()` and `toString()`

You can also specify your choice of format in the parameters of the `DateObject.toLocaleString()` function.

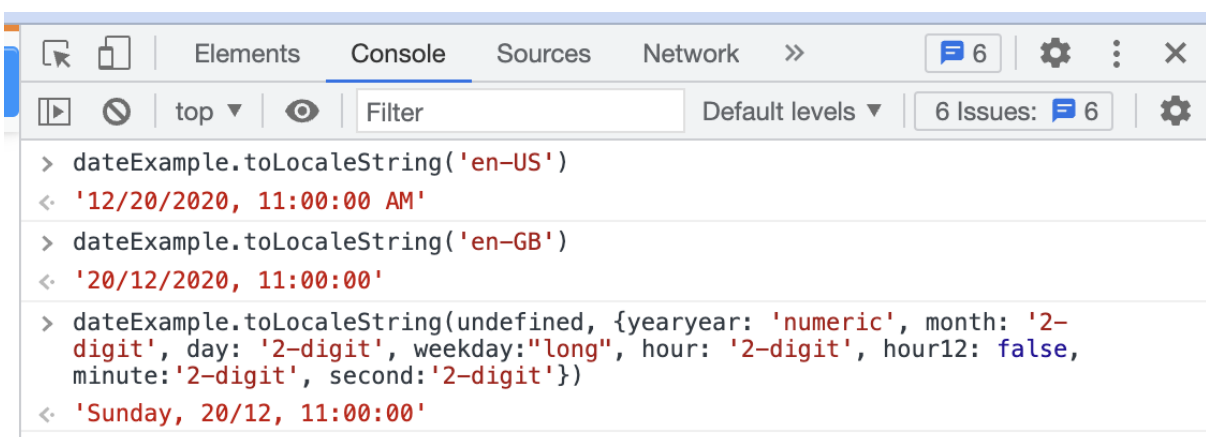


Figure 15. Change date format with a different style or customized styles.

Exercise 2:

Create a date string of the current date and time in three different formats: "Day-Month-Year, HH:MM:SS AM/PM", "Year-Month-Day, HH:MM:SS", and "Weekday, Day/Month, HH:MM:SS".

3. Adding Interactivity with JavaScript

3.1 Variables in JavaScript

Variables in JavaScript can be declared with the **var** keyword, as shown in the example in Figure 16. 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 camelcase 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](#) (section 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 16. A simple example of declaring and using variables

3.2 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 17. 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 an `` 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 17,

- 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 `` element to demonstrate click events getting triggered on the click of the button. Add a `numberOfClicks` attribute to the `` 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 20 is called.
- Just before closing the `<body>` element, add an `<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 17. 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 element.

```

#counter {
  margin-left: 30px;
}

```

Figure 18. 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 19. By doing this, it is easy to separate function declarations and function calls.

```

JS events.js
1  ✓ // Function declarations
2
3  // Script calls|

```

Figure 19. 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 17). 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 20 shows the entire function. Make sure you understand each step in this

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

Figure 20. 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 21).

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

Figure 21. 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 `<div>` element as a container to hold an `` element. Add the path of the `toggle.png` image file into the `src` attribute of the image element, as shown in Figure 22. 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>
  
</div>
```

Figure 22. 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 23).

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

Figure 23. `events.css` file

- In the `events.js` file, create the function named `toggleVisibility()`, which will be the event handler. Each 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 24. Toggle Visibility function

- As shown in Figure 24, the variable `chartImage` selects 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 24 compares the element's display property with "none" and "block" strings. The code after the IF checks the `style.display` property to `block` (if the element is currently hidden), else `style.display` is set to `none`.