MacEwan UNIVERSITY | SCHOOL OF CONTINUING EDUCATION

Introduction to JavaScript

CMSK 0152

Course Manual

Course Manual

2021. Version 1.

Author(s): Hammad Tawfig
Editor(s): Shanna Rowney
Reviewer(s): Anastasia Maywood

Cover Photo: tsmr. (2018). CC0. Retrieved from https://pixabay.com/photos/code-javascript-program-source-code-3337044/

TABLE OF CONTENTS

INTRODUCTION

Course Overview

In this course, you will be introduced to JavaScript. You will learn how to create valuable functions using basic syntax, arrays, and objects. Moreover, you will explore how to program in JavaScript, debug JavaScript, and integrate JavaScript code into HTML pages to make them more dynamic and interactive.

There are five modules in the course:

1. Introduction to JavaScript
2. JavaScript Variables, Data Types, and Functions
3. JavaScript Objects, Arrays, Operators, and Conditions
4. JavaScript Loops, Promises, and Async
5. JavaScript Exercise-Based Practice

Initial Thoughts

Before beginning the course, take some time to reflect on the topic of web development. You are not required to write or submit these thoughts anywhere.

In your own words, what is JavaScript?
Why do you think understanding JavaScript is valuable?
What do you hope to learn in this course?

COURSE LEARNING OUTCOMES

After completion of this course, you will be able to:

1. **Describe JavaScript basics and the Document Object Model (DOM)**
2. **Review JavaScript best practices such as variable naming rules**
3. **Employ JavaScript objects and arrays usages**
4. **Employ JavaScript functions and methods**
5. **Integrate JavaScript in HTML web pages**
6. **Manipulate HTML web page basic events with JavaScript**

Self Assessment

Setting goals is an important step when you start anything new – such as this course. SMART goals are those that are specific, measurable, attainable, relevant, and timely. You are more likely to achieve SMART goals. Please take some time to think of one or two goals that you would like to achieve by the time you complete this course. Then, consider each goal with regard to the SMART divisions below.

S	Make it specific	What do you want to accomplish?
M	Make it measurable	How will you know when you have accomplished your goal?
A	Make it attainable	How can the goal be accomplished?
R	Make it relevant	Is this goal worth working hard to accomplish?
T	Make it timely	By when will the goal be accomplished?

MODULE 1 – INTRODUCTION TO JAVASCRIPT

Overview

In this module you will gain a basic understanding of what JavaScript is, what it is capable of, and how it could be used in a website. You will also review what the Document Object Model is and how it fits in a website.

LEARNING OUTCOMES

After completion of this module, you will be able to:

1. **Define JavaScript language**
2. **Identify what can be accomplished with JavaScript**
3. **Identify the tools that will be used in learning JavaScript throughout the course**
4. **Code and execute a simple "Hello World" program**
5. **Change the content of HTML elements using JavaScript**
6. **Add and delete HTML elements using JavaScript**

Lesson 1: JavaScript Introduction

What is JavaScript?

JavaScript is a client-side programming language with a lot of capability (MDN Web Docs, n.d.). The language was first known as LiveScript before being renamed JavaScript.

JavaScript is mostly used to improve the user's engagement with a web page. In other words, you may use JavaScript to make your website more vibrant and engaging. JavaScript is also commonly utilised in the production of games and mobile applications. JavaScript's grammar is heavily influenced by the programming language C.

Many programmers mistakenly believe that JavaScript and Java are the same thing. In fact, JavaScript and Java have little in common. Java is a powerful programming language, but JavaScript is a simple scripting language.

How to Run JavaScript

JavaScript cannot run on its own because it is a scripting language. The browser is in charge of executing JavaScript code. When a user requests an HTML page that contains JavaScript, the script is transmitted to the browser, and the browser is responsible for executing it (Hartman, 2021). The key benefit of JavaScript is that it is supported by all modern web browsers.

As a result, you don't have to be concerned about whether your site visitor is using Internet Explorer, Google Chrome, Firefox, or another browser. JavaScript will be available. Furthermore, JavaScript is compatible with every operating system, including Windows, Linux, and Mac. As a result, JavaScript solves the primary drawbacks of VBScript (now obsolete), which is confined to just Internet Explorer and Windows.

What Tools Do You Need?

To begin, you'll need a text editor to write your code and a browser to see the web pages you create. You can use whatever text editor you choose, such as Notepad++, Visual Studio Code, Sublime Text, Atom, or any other text editor you are familiar with. Any online browser, including Google Chrome, Firefox, Microsoft Edge, and Internet Explorer, can be used.

Ready to Code Your First JavaScript JavaScript Program?

You should place all your JavaScript code within "<script>" tags. if you are keeping your JavaScript code within the HTML document itself (MDN Web Docs, n.d.-a). This helps your browser distinguish your JavaScript code from the rest of the code.

Because there are alternative client-side scripting languages (i.e., VBScript), it is strongly advised that you indicate the scripting language you intend to utilise. You must include the type attribute within the <script> element and set its value to text/javascript, as seen below:

```
<script type="text/javascript">
  // JavaScript code here
</script>
```

Hello World Example:

```
<html>
  <head>
    <title>This is my first web application with JavaScript.</title>

    <script type="text/javascript">
      alert("Hello World!");
    </script>
  </head>

  <body></body>
</html>
```

Summary

JavaScript is a programming language that is mostly used on the client side, in the browser. JavaScript has a lot of power and virtually limitless open-source resources. Before being renamed JavaScript, the language was known as LiveScript. Because it is supported by all contemporary online browsers, you don't have to worry about your site visitors using a different browser.

In-Class Exercise 1

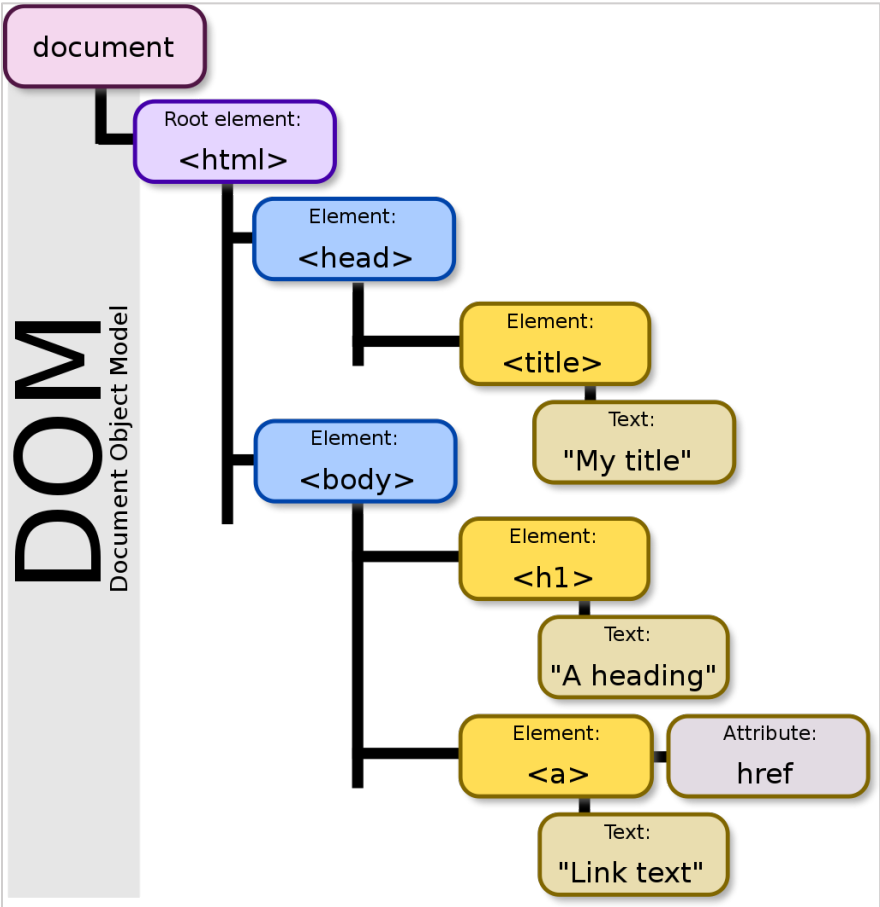
1. What is JavaScript?
2. Explain the difference between Java and JavaScript.

Lesson 2: Document Object Model (DOM) Introduction

What is the DOM?

The DOM is a W3C (World Wide Web Consortium) standard defined as, "The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document" (W3 Schools, n.d.-a, para. 7).

The HTML DOM is constructed as a tree of Objects:



Birger Eriksson. (2012). CC BY-SA 3.0. Found here.

What are the W3C DOM Standards?

The W3C DOM standard is two parts. The first is DOM Core which is what is used for XML documents. The second part is the DOM HTML which holds all the definitions for HTML documents.

What is the HTML DOM?

HTML DOM is the standard object model and programming interface for HTML (W3 Schools, n.d.-a). It is the standard on how to get, change, add or remove HTML elements (n.d.-a).

What are some of the things you could accomplish with DOM?

With the DOM, JavaScript can change/remove/add/ all the HTML and CSS elements and attributes on the page (W3 Schools, n.d.-a).

Changing HTML Content

Here is how you can use the DOM to modify a paragraph HTML element <p>:

```
<html>
  <body>

    <script>
      document.getElementById("paragraph1").innerHTML = "Hi All!";
    </script>

    <p id="paragraph1">Hello to all the world!</p>

  </body>
</html>
```

Here is another example of how you may use the DOM to modify a heading HTML element <h1>:

```
<html>
  <body>
    <script>
      const element = document.getElementById("heading1");
      element.innerHTML = "Today";
    </script>
    <h1 id="heading1">Yesterday</h1>
  </body>
</html>
```

Changing the Value of an Attribute

With the DOM, you can use JavaScript on the values of HTML attributes. Let us consider an example where you have an HTML page that displays an image of a *sad face* emoji, and you would like to use JavaScript to switch the image to a *happy face* emoji:

```
<!DOCTYPE html>
<html>
  <body>
    <img id="image1" src="sad.gif" />

    <script>
      document.getElementById("image1").src = "happy.jpg";
    </script>
  </body>
</html>
```

Dynamic HTML Content

With JavaScript, you can display dynamic content on your HTML page. Consider an example where you would like your HTML page to display the current date and time every time the page is loaded or refreshed:

```
<!DOCTYPE html>
<html>
  <body>
    <p id="paragraph1"></p>

    <script>
      document.getElementById("paragraph1").innerHTML = "Today's Date and
Time : " + Date();
    </script>
  </body>
</html>
```

Summary

With the DOM, JavaScript can change/remove/add/ all the HTML and CSS elements and attributes on a page.

In-Class Exercise 2

1. Write JavaScript that displays the date and time a given document was last modified.

```
<!DOCTYPE html>
<html>
  <body>
    <p>This document was last modified <span id="span1"></span>.</p>

    <script>
      // Your answer goes here
    </script>
  </body>
</html>
```

2. Write JavaScript that will change a “We Are Closed” image to a “We Are Open” image. *Note:* The “We Are Open” image file on the server is named: we_are_open.gif

```
<!DOCTYPE html>
<html>
  <body>
    <span id="span1"> <img src="we_are_closed.gif" /> </span>

    <script>
      // Your answer goes here
    </script>
  </body>
</html>
```

3. What are some of the different ways to get an element from DOM?

MODULE 2 – JAVASCRIPT VARIABLES, DATA TYPES, AND FUNCTIONS

Overview

In this module, you will gain experience writing JavaScript, develop a basic understanding of what writing a JavaScript program involves, and explore how to utilize variables and functions.

LEARNING OUTCOMES

After completion of this module, you will be able to:

1. **Describe JavaScript variables**
2. **Describe JavaScript variable naming conventions**
3. **List different JavaScript data types**
4. **Explain the purpose of different JavaScript data types**
5. **Describe the purpose of JavaScript functions**
6. **Describe the syntax of JavaScript functions**
7. **Describe JavaScript functions naming conventions**
8. **Code simple JavaScript Programs utilizing different variables and functions.**

Lesson 1: JavaScript Variables

What is a JavaScript Variable?

"A JavaScript variable is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable" (JavaTpoint, n.d., para. 1).

Declare Variables in JavaScript

Before using a variable, you must first declare it. You must use the *var* keyword to declare variables. For example:

```
var lastname;
var firstName;
// OR you could declare multiple in one line
var lastname, firstName;
```

Assign a Value to the Variable

You can assign a value to the variable when you declare it. You may also declare the variable first and then assign a value to it some other place in your code. Below is an example of each method:

```
var lastname = "Smith";

// OR

var lastname;
lastname = "Smith";
```

Naming Variables

Though you are free to call the variables whatever you like, it is common programming practise to give them descriptive and meaningful names. In addition, variable names must begin with a letter and are case sensitive. For example: The variables *teachername* and *teacherName* are different.

Variables Examples

Here some examples of variables and how they may be used:

```
<html>
  <head>
    <title>Variables Examples</title>

    <script>
      var x = 4;
      var y = 2;
      // We can do math with the variables.
      var addition = x + y;
      var subtraction = x - y;
      var multiplication = x * y;
      var division = x / y;

      // Then, we can display:
      document.write(" x + y = " + addition + "<br/>");
      document.write(" x - y = " + subtraction + "<br/>");
      document.write(" x * y = " + multiplication + "<br/>");
      document.write(" x / y = " + division + "<br/>");
    </script>
  </head>

  <body></body>
</html>
```

Reuse or Create?

Some programmers prefer to reuse existing variables rather than declare new ones. As a result, their variables are like a previously organized, labelled box that has had various different items dropped into it. Some items match the original label, and some do not. What's really inside the box? We don't know for sure and often we have to take a closer look at it to figure it out properly. This is called debugging.

These such programmers save a little time on variable declaration but lose 10x as much time debugging. Simply adding an extra variable can save a lot of time in the end.

Modern JavaScript minifiers and browsers optimise code sufficiently enough that performance issues aren't a concern. Using distinct variables for different values can also aid the optimization of your code by the engine.

Summary

Declaring variables is how we store data in JavaScript by using the *var*. We should name our variables in a way that makes it easy for any other programmer who might look at them to understand what they were declared for.

In-Class Exercise 1

1. Using JavaScript:
(1) Declare a variable
(2) Name it *candy*
(3) Assign it a value of *TWIX*
(4) Display the value of the variable in an alert.

```
<html>
  <head>
    <script>
      // Your answer goes here

    </script>
  </head>
  <body></body>
</html>
```

2. Using JavaScript:
(1) Declare a variable
(2) Name it *y*
(3) Assign it a value of *55*
(4) Display the value of the variable in an alert.

```
<html>
  <head>
    <script>
      // Your answer goes here

    </script>
  </head>
  <body></body>
</html>
```

3. Using JavaScript:
(1) Show the sum of 3 + 10
(2) Display the sum in an alert
(3) Use two variables (*x* and *y*) to calculate the sum value

```
<html>
  <head>
    <script>
      // Your answer goes here

    </script>
  </head>
  <body></body>
</html>
```

Lesson 2: JavaScript Data Types

What are Data Types?

In JavaScript, a value is always of a specific type – a string or a number, for example. “Data types basically specify what kind of data can be stored and manipulated within a program” (Tutorial Republic, n.d., para. 1).

There are 3 categories of data types:

1. Primitive:
    a. String
    b. Number
    c. Boolean

2. Special:
    a. Null
    b. Undefined

3. Reference:
    a. Object
    b. Array
    c. Function

The String Data Type

A JavaScript string stores a series of characters like "John Doe". A string can be any text inside double or single quotes:

```
let carName1 = "Volvo XC60";
let carName2 = 'Volvo XC60';
```

String indexes are zero-based: The first character is in position 0, the second in 1, and so on. (W3 Schools, n.d.-b, para. 1-4)

Here are some examples:

```
<html>
  <head>
    <script type="text/javascript">
      var text1 = 'This was declared with single quotes';
      var text2 = "This was declared with double quotes";
      var text3 = "This was declared with a 'single quote' in double
quotes";
      var text3 = 'This was declared with a "double quote" in single
quotes';
    </script>
  </head>

  <body></body>
</html>
```

The Number Type

> The JavaScript Number type is a double-precision 64-bit binary format IEEE 754 value, like double in Java or C#. This means it can represent fractional values, but there are some limits to what it can store. A Number only keeps about 17 decimal places of precision; arithmetic is subject to rounding. (MDN Web Docs, n.d.-b, para. 3)

For example:

```
<html>
  <head>
    <script>
      // Declaring some numbers
      var anInteger = 55;
      var aDouble = 2.5;

      // This is not a number; why?
      alert("1" / 44 + 33);
    </script>
  </head>

  <body></body>
</html>
```

The Boolean Data Type

There are only two possible values for the Boolean data type: true or false. As seen below, it is commonly used to hold values like:

- yes (true) or no (false)
- on (true) or off (false)
- and so on

Let us consider the following examples:

```
<html>
  <head>
    <script>
      var isMikeHere = true;
      var isJohnHere = false;

      // OR you can do some comparisons

      var two = 2;
      var three = 3;
      var four = 4;

      alert(three > two); // What is the output?
      alert(three > four); // What is the output?
    </script>
  </head>

  <body></body>
</html>
```

The Undefined Data Type

Only the special value *undefined* can be assigned to the undefined data type. The value of an undefined variable is one that has been declared but not assigned a value.

Let us consider the following example:

```
<html>
  <head>
    <script>
      var text1;
      var text2 = "I am going to be a good programmer.";

      alert(text1); // What is the output?
      alert(text2); // What is the output?
    </script>
  </head>

  <body></body>
</html>
```

The Null Data Type

“This is another special data type that can have only one value-the null value. A null value means that there is no value. It is not equivalent to an empty string ("") or 0, it is simply nothing” (Tutorial Republic, n.d., para. 16).

Let us consider the following example:

```
<html>
  <head>
    <script>
      var aValue = null;
      alert(aValue); // What is the output?

      var aValue = "a new value";
      alert(aValue); // What is the output?

      aValue = null;
      alert(aValue); // What is the output?
    </script>
  </head>

  <body></body>
</html>
```

The Object Data Type

The object is a complex data type that allows you to store collections of data.

An object contains properties, defined as a key-value pair. A property key (name) is always a string, but the value can be any data type, like strings, numbers, booleans, or complex data types like arrays, function and other objects.

The following example will show you the simplest way to create an object in JavaScript.

```
<html>
  <head>
    <script>
      var emptyObject = {};
      var person = { name: "Clark", surname: "Kent", age: "36" };

      var car = {
        modal: "BMW X3",
        color: "white",
        doors: 5,
      };
    </script>
  </head>

  <body></body>
</html>
```

(Tutorial Republic, n.d., para. 18-19)

The Array Data Type

An array is a type of object used for storing multiple values in single variable. Each value (also called an element) in an array has a numeric position, known as its index, and it may contain data of any data type-numbers, strings, booleans, functions, objects, and even other arrays. The array index starts from 0, so that the first array element is arr[0] not arr[1].

The simplest way to create an array is by specifying the array elements as a comma-separated list enclosed by square brackets, as shown in the example below:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      var colors = ["Red", "Yellow", "Green", "Orange"];
      var cities = ["London", "Paris", "New York"];

      alert(colors[0]); // What is the output?
      alert(cities[2]); // What is the output?
    </script>
  </head>

  <body></body>
</html>
```

(Tutorial Republic, n.d., para. 20-21)

The Function Data Type

The function is [a] callable object that executes a block of code. Since functions are objects, [...] it is possible to assign them to variables, as shown in the example below:

```
<html>
  <head>
    <script>
      var morningGreeting = function () {
        return "Good morning!";
      };

      // Let us check the type of morningGreeting.
      alert(typeof morningGreeting); // What is the output?

      // Now, let us call the function.
      alert(morningGreeting()); // What is the output?
    </script>
  </head>

  <body></body>
</html>
```

In fact, functions can be used at any place any other value can be used. Functions can be stored in variables, objects, and arrays. Functions can be passed as arguments to other functions, and functions can be returned from functions. Consider the following function:

```
<html>
  <head>
    <script>
      function makeMorningGreetingFor(aName) {
        return "Good morning, " + aName;
      }

      alert(makeMorningGreetingFor("Moe")); // What is the output?
    </script>

  <body></body>
</html
```

(Tutorial Republic, n.d., para. 22-23)

The typeof Operator

The typeof operator can be used to find out what type of data a variable or operand contains. It can be used with or without parentheses (typeof(x) or typeof x).

The typeof operator is particularly useful in the situations when you need to process the values of different types differently, but you need to be very careful, because it may produce unexpected result in some cases, as demonstrated in the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      // Numbers
      alert(typeof 44);
      alert(typeof 23.6);

      // Strings
      alert(typeof "");
      alert(typeof "hi");
      alert(typeof "44534345");

      // Booleans
      alert(typeof true);
      alert(typeof false);

      // Undefined
      alert(typeof undefined);

      // Null
      alert(typeof Null);

      // Objects
      alert(typeof { firtName: "Mike", age: 34 });

      // Arrays
      alert(typeof [5, 5, 7, 3]);

      // Functions
      alert(typeof function () {});
    </script>
  </head>

  <body></body>
</html>
```

(Tutorial Republic, n.d., para. 24-25)

In-Class Exercise 2

1. Which of the following is/are strings? (Select all that apply.)
   a. 1
   b. 2.0
   c. '3'

2. Which of the following is/are numbers? (Select all that apply.)
   a. 2
   b. 5.0
   c. '9'
   d. -6

3. Which of the following is/are Booleans? (Select all that apply.)
    a. true
    b. false
    c. "False"

4. What is the result of 8 + "8"? (Select the correct answer.)
    a. 16
    b. "16"
    c. "88"

5. What is the result 444 + 888? (Select the correct answer.)
    a. number
    b. string

Lesson 3: JavaScript Functions

What is a Function?

> Functions (also called 'procedures' in some programming languages and 'methods' in most object oriented programming languages) are a set of instructions bundled together to achieve a specific outcome. Functions are a good alternative to having repeating blocks of code in a program. (Future Learn, n.d., para. 1).

How to Define Functions in JavaScript

> A function in JavaScript is similar to a procedure—a set of statements that performs a task or calculates a value - but for a procedure to qualify as a function, it should take some input and return an output where there is some obvious relationship between the input and the output. (MDN Web Docs, n.d.-c, para. 1).

Parameters and arguments are nearly identical, with just a little distinction between them. Arguments are the values supplied to the function when it is called to be performed; whereas parameters are used when constructing a function.

To call a function, use its name, followed by a pair of parentheses, and any parameters if they exist. You can also save the results of calling a function in a variable or constant.

Let us consider the following example:

```
<html>
  <head>
    <script>
      // How to declare a function
      function addXandY(x, y) {
        return x + y;
      }

      var value = addXandY(4, 4);

      // Many times we do not want to display the value in alerts
      // This is a different way you will find beneficial in debugging in
the future
      console.log(value);
```

```
      // You could also use the keyword "function" to declare a function
      const subtractXFromY = function (x, y) {
        return y - x;
      };

      // You could call the function directly in you consule.log statement
      console.log(subtractXFromY(4, 7));

      // You could also use an arrow function expression to declare a
function
      const multiplyXByY= (x, y) => {
        return x * y;
      };

      // And the result is
      console.log(multiplyXByY(4, 7));
    </script>
  </head>
  <body></body>
</html>
```

The return keyword gave the function's return value to whatever invoked the function (in this instance, the variable result). If you use several return statements in the same function, only the first one will be performed because, when a controls encounters such a statement, it immediately exits the current function and returns the result to the code that requested the function.

Functions Can Declare Their Own Variables or Use Outer Once

Functions require additional variables to complete their tasks properly. Therefore, we must declare these variables within our functions and use them to get the desired outcome. These variables declared within the function (between the curly brackets) are known as local variables, and they are only accessible within that function; we cannot access or utilise them in any other code.

Let us consider this example:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var name = "Mike"; // This is an outer variable

      function GoodMorning() {
        let greetings = "Good Morning!" + name; // This is a local
variable

        console.log(greetings);
      }

      GoodMorning();

      // Local variables cannot be read from outside the function;
therefore, the following will error out
      console.log(greetings);
```

```
    </script>
  </head>
  <body></body>
</html>
```

Summary:

- The values given to the function are referred to as arguments
- We have seen how a function can access and modify external variables
- Functions can contain local variables; however, these local variables are not accessible outside the function body
- If a function does not have a return value, it will return *undefined*

In-Class Exercise 3

1. You want to add a few numbers.
   (1) Declare a function that adds 3 numbers.
   (2) This function must add 2 of the 3 numbers using another function that adds 2 numbers.
   (3) Use your function to display the result of 5+4+6 in the console window.

```
      <html>
  <head>
    <script>
      // Your answer goes here


    </script>
  </head>
  <body></body>
</html>
```

MODULE 3 – JAVASCRIPT OBJECTS, ARRAYS, OPERATORS, AND CONDITIONS

Overview

In this module, you will gain experience with operators and assignments to manipulate arrays and perform basic math operations.

LEARNING OUTCOMES

After completion of this module, you will be able to:

1. **Describe JavaScript objects**
2. **Define JavaScript arrays**
3. **Identify JavaScript addition, subtraction, multiplication, and division operators**
4. **Identify JavaScript addition, subtraction, multiplication, and division assignments**
5. **Explain JavaScript conditional statements and operators**
6. **Code simple JavaScript Programs utilizing different arrays and operations**

Lesson 1: JavaScript Objects and Arrays

What is an Object?

In object-oriented programming (OOP), objects are the things you think about first in designing a program and they are also the units of code that are eventually derived from the process. In between, each object is made into a generic class of object and even more generic classes are defined so that objects can share models and reuse the class definitions in their code. Each object is an instance of a particular class or subclass with the class's own methods or procedures and data variables. (Tech Target, 2005a, para. 1)

What are JavaScript Objects?

The following is paraphrased from MDN Web Docs (n.d.-d):

Objects in JavaScript, like things in many other programming languages, may be compared to real-world objects. A JavaScript object is connected with properties. An object's property may be thought of as a variable that is connected to the object. Except for the attachment to objects, attributes are the same as regular JavaScript variables.

Objects are made up of attributes. If the property includes the function, it is considered the object's method. Otherwise, the characteristic is considered to be property. The attributes of an item define the thing's qualities. The following basic dot-notation is used to access an object's properties.

Here is an example of a JavaScript object:

```
<html>
  <head>
    <script>
      let Computer = new Object();
      Computer.type = "PC";
      Computer.os = "Windows";
      Computer.memory = "16GB";

      console.log(Computer);
```

```
    </script>
  </head>
  <body></body>
</html>
```

You can also get the property values that you need:

```
<html>
  <head>
    <script>
      let Computer = new Object();
      Computer.type = "PC";
      Computer.os = "Windows";
      Computer.memory = "16GB";

      console.log(Computer.type);
      console.log(Computer.os);
    </script>
  </head>
  <body></body>
</html>
```

You can also get the values of the properties using the brackets notation:

```
<html>
  <head>
    <script>
      let Computer = new Object();
      Computer.type = "PC";
      Computer.os = "Windows";
      Computer.memory = "16GB";

      console.log(Computer["type"]);
      console.log(Computer["os"]);
      console.log(Computer["memory"]);
    </script>
  </head>
  <body></body>
</html>
```

You can also declare a method. A method is a function declared in an object:

```
<html>
  <head>
    <script>
      let Computer = {
        type: "PC",
        os: "Windows",
        memory: "16GB",
        specs: function () {
          return (
            "This a " +
            this.os +
            " " +
            this.type +
            " with " +
            this.memory +
            " of memory."
          );
        },
      };
      console.log(Computer.specs());
    </script>
  </head>
  <body></body>
</html>
```

You can also declare an object by calling a function that creates the object for you like the example below:

```
<html>
  <head>
    <script>
      function Computer(type, os, memory) {
        this.type = type;
        this.os = os;
        this.memory = memory;
      }

      let myComputer = new Computer("Mac", "macOS", "32GB");

      console.log(myComputer.type, myComputer.os, myComputer.memory);
    </script>
  </head>
  <body></body>
</html>
```

What is an Array?

“An array is a series of memory locations – or ‘boxes’ – each of which holds a single item of data, but with each box sharing the same name. All data in an array must be of the same data type” (BBC, n.d., para. 1). The items in an array may be accessed by referring to their index number, and the index of the array's first member is zero.

Here is how we can create an array:

```
<html>
  <head>
    <script>
      var students = ["John", "Ann", "Jane", "Moe"];
    </script>
  </head>
  <body></body>
</html>
```

Here is how we can access the values in an array:

```
<html>
  <head>
    <script>
      var students = ["John", "Ann", "Jane", "Moe"];

      alert(students[0]); //What is the output?
    </script>
  </head>
  <body></body>
</html>
```

Here is how we can change a value in an array:

```
<html>
  <head>
    <script>
      var students = ["John", "Ann", "Jane", "Moe"];

      students[0] = "Nav"

      alert(students[0]); //What is the output?
    </script>
  </head>
  <body></body>
</html>
```

Here is how we can use an array constructor to create an array:

```
<html>
  <head>
    <script>
      var students = new Array("John", "Ann", "Jane", "Moe");

      alert(students[0]); //What is the output?

      // or

      var teachers = new Array();

      teachers[0] = "Mikky";
      teachers[1] = "Dani";
      teachers[2] = "Ahmad";

      alert(teachers[2]); //What is the output?
    </script>
  </head>
  <body></body>
</html>
```

JavaScript Array Methods

The array object contains many attributes and methods that allow developers to quickly and effectively handle arrays. Here is a list of these methods: JavaScript Array Reference

Let us examine the following example:

```
<html>
  <head>
    <script>
      var students = new Array("John", "Ann", "Jane", "Moe");

      Array.prototype.outputToConsole = function () {
        for (i = 0; i < this.length; i++) {
          console.log([i]);
        }
      };

      students.outputToConsole();
      console.log("This how many students in the array: ",
students.length);

      students.sort();
      console.log("Sorted: ");
      students.outputToConsole();

      students.reverse();
      console.log("Reversed: ");
      students.outputToConsole();
```

```
      students.pop();
      console.log("One student removed: ");
      students.outputToConsole();

      students.push("David");
      console.log("New student added: ");
      students.outputToConsole();
    </script>
  </head>
  <body></body>
</html>
```

In-Class Exercise 1

1. Refer to the below code:
   (1) Write a function that outputs the length of the array to *console.log*. Name the function *brandsCount*
   (2) Call *brandsCount*
   (3) Write another function that returns "true" if there are more then ten brands in the array; otherwise, it returns "false".

```
<html>
  <head>
    <script>
      var brands = ["Apple", "Samsung", "Mercedes", "Burger King"];
    </script>
  </head>
  <body></body>
</html>
```

Lesson 2: JavaScript Operators and Assignments

What are JavaScript Operators?

> JavaScript includes operators, same as other languages. An operator performs some operation on single or multiple operands (data value) and produces a result. For example, in 1 + 2, the + sign is an operator and 1 is left side operand and 2 is right side operand. The + operator performs the addition of two numeric values and returns a result. (Tutorials Teacher, n.d., para. 1-2)

The JavaScript categories of operators that we will explore are:

1. Arithmetic Operators
2. String Concatenation
3. Comparison Operators
4. Logical Operators
5. Assignment Operators
6. Ternary Operators

Arithmetic Operators

A list of all arithmetic operators can be found here: JavaScript Operators Reference

Here are some examples:

```
<html>
  <head>
    <script>
      var one = 1;
      var two = 2;

      var value = one + two;
      console.log(value);

      value = two - one;
      console.log(value);
      value = one * two;
      console.log(value);
      value = two / one;
      console.log(value);
      value = one % 2;
      console.log(value);
    </script>
  </head>
  <body></body>
</html>
```

“The ++ and -- operators are unary operators. It works with either left or right operand only. When used with the left operand, e.g., x++, it will increase the value of x when the program control goes to the next statement” (Tutorials Teacher, n.d., para. 5)

Let us consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var counter = 1;

      counter++;
      console.log(counter);
      ++counter;
      console.log(counter);
      counter--;
      console.log(counter);
      --counter;
      console.log(counter);
    </script>
  </head>
  <body></body>
</html>
```

String Concatenation

A list of all string operators can be found here: JavaScript Operators Reference

Let us consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var six = 6;
      var hiThere = "Hi There ";
      var name = "Mike";
      var ten = 11;

      console.log(six + hiThere);
      console.log(hiThere + name);
      console.log(six + ten);
      console.log(hiThere + true);
      console.log(name - hiThere);
    </script>
  </head>
  <body></body>
</html>
```

Comparison Operators

A list of all comparison operators can be found here: JavaScript Operators Reference

Let us consider the following example:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      var six = 6;
      var five = 5;
      var ten = "10";
      var anotherSix = six;

      console.log(six == ten);
      console.log(six === ten);
      console.log(six == anotherSix);
      console.log(six != five);
      console.log(six > five);
      console.log(six < five);
      console.log(six >= five);
      console.log(six <= five);
    </script>
  </head>
  <body></body>
</html>
```

Logical Operators

A list of all logical operators can be found here: JavaScript Operators Reference

Let us consider the following example:

```
<html>
  <head>
    <script>
      var four = 4;
      var nine = 9;

      console.log(four < nine || four == nine);
      console.log(!(four < nine));
      console.log(!(four > nine));
      console.log(four != nine && a < nine);
      console.log(four > nine || four == nine);
    </script>
  </head>
  <body></body>
</html>
```

Assignment Operators

A list of all assignment operators can be found here: JavaScript Operators Reference

Let us consider the following example:

```
<html>
  <head>
    <script>
      var four = 4;
      var three = 3;
      var value = four;

      value = three;
      console.log(value);

      value += 1;
      console.log(value);

      value -= 1;
      console.log(value);

      value *= 5;
      console.log(value);

      value /= 5;
      console.log(value);

      value %= 2;
      console.log(value);
```

```
    </script>
  </head>
  <body></body>
</html>
```

Ternary Operators

"JavaScript provides a special operator called ternary operator :? that assigns a value to a variable based on some condition. This is the short form of the *if else* condition" (Tutorials Teach, n.d., para. 11).

For more information about the ternary operator, visit here: JavaScript Operators Reference

Let us consider the following example:

```
<html>
  <head>
    <script>
      var two = 2;
      var one = 1;

      var value1 = two > one ? two : one;
      console.log(value1);

      var value2 = one > two ? two : one;
      console.log(value2);
    </script>
  </head>
  <body></body>
</html>
```

In-Class Exercise 2

1. Complete the following to the below console log using JavaScript operators. Use "addition" as an example of how to complete the remaining console log statements:
   (1) Multiply 4 with 6, alerting the result.
   (2) Alert the reminder, once 12 is divided by 5.

```
<html>
  <head>
    <script>
      let five = 5;
      let three = 3;

      // addition
      console.log("five + y = ", five + three); // 8

      // subtraction
      // console.log 2
```

```
      // multiplication
      // console.log 15

      // division
      // console.log 1.6666666666666667

      // remainder
      // console.log 2
    </script>
  </head>
  <body></body>
</html>
```

MODULE 4 – JAVASCRIPT LOOPS, PROMISES, AND ASYNC

Overview

In this module, you will experience writing JavaScript Loops and Asynchronous functions.

LEARNING OUTCOMES

After completion of this module, you will be able to:

1. **List different JavaScript loop types**
2. **Describe how to run functions in parallel**
3. **Describe the purpose of JavaScript promises**
4. **Code simple JavaScript programs utilizing different loops and asynchronous functions**

Lesson 1: JavaScript Loops

What is a Loop?

> In computer programming, a loop is a sequence of instructions that is continually repeated until a certain condition is reached. Typically, a certain process is done, such as getting an item of data and changing it, and then some condition is checked such as whether a counter has reached a prescribed number. (Tech Target, 2005b, para. 1)

A loop is a section of code that will be repeated indefinitely. Loops are classified into two types: "while loops" and "for loops." *While* loops will continue to repeat until a condition is no longer true; *for* loops will stop after a certain number of repetitions.

Why do I need Loops?

Loops come in handy when you need to execute the same lines of code repeatedly for a set number of times or as long as a certain condition is true.

Assume you wish to input the word "Hi" 99 times on your website. You could copy and paste the identical line 99 times. Or, instead, if you utilise loops, you can do this work in as little as 4 or 5 lines.

What are the Different Sub-types of Loops in JavaScript?

There are four sub-types of loops in JavaScript that we will explore here:

1. *for* loops
2. *for/in* loop
3. *while* loop
4. *do...while* loop

For Loop

Here is the syntax:

```
<html>
  <head>
    <script>
      for (statement1; statement2; statement3) {
        // The code you want
      }
    </script>
```

```
  </head>
  <body></body>
</html>
```

Let us use the *for* loop to output the names of students in HTML, as an example:

```
<html>
  <head>
    <script>
      var students = new Array("Zoe", "Jane", "Adam", "Elizabeth",
"Anna");

      document.write("<b>This is being done using for loops </b><br />");

      for (i = 0; i < students.length; i++) {
        document.write(students[i] + "<br />");
      }
    </script>
  </head>
  <body></body>
</html>
```

While Loop

Here is the syntax:

```
<!DOCTYPE html>
<html>
  <head>
    <script type="text/javascript">
      while (condition) {
        // write some code here
      }
    </script>
  </head>
  <body></body>
</html>
```

One very important thing about *while* loops, is that they run as long as the condition is true. Not managing loops properly could crash your application and create memory leaks.

You should include a phrase that will stop the loop at some point inside the *while* loop. Otherwise, your loop will never be completed and your browser may crash.

Here is an example:

```
<html>
  <head>
    <script>
      var index = 0;
      var max = 100;
```

```
      document.write("<b>This is being done using while loops </b><br
/>");
      document.write(
        "This is outputting each number from " +
          index +
          " up to " +
          max +
          "<br />"
      );

      while (index <= max) {
        document.write(index + "<br />");

        index++;
      }
    </script>
  </head>
  <body></body>
</html>
```

Do While Loop

Here is the syntax:

```
<html>
  <head>
    <script type="text/javascript">
      do {
        // write some code here
      } while (condition);
    </script>
  </head>
  <body></body>
</html>
```

The *do...while* loop is similar to the *while* loop. The main difference is that in a *do...while* loop, the block of code is executed before the condition is checked.

Here is an example:

```
<html>
  <head>
    <script>
      var index = 0;
      var max = 100;

      document.write("<b>This is being done using while loops </b><br
/>");
      document.write(
        "This is outputting each number from " +
          index +
          " up to " +
```

```
          max +
          "<br />"
      );

      do {
        document.write(index + "<br />");

        index++;
      } while (index <= max);
    </script>
  </head>
  <body></body>
</html>
```

In-Class Exercise 1

1. What will the output of the following code be? (Select the correct answer.)
   a. The numbers in the array, in order, to console.log
   b. The numbers in the array, in the reverse order, to console.log
   c. 0, the length of the array
   d. "The array is empty!"

```
<html>
  <head>
    <script>
      function logArray(arrayOfThings) {
        var length = arrayOfThings.length,
          i = 0;
        if (length == 0) console.log("The array is empty!");
        else {
          do {
            console.log(arrayOfThings[i]);
          } while (++i < length);
        }
      }

      let numbers = new Array(1, 2, 4, 5, 7, 8, 10);

      logArray(numbers);
    </script>
  </head>
  <body></body>
</html>
```

Lesson 2: JavaScript Async

What is Async/Await?

"Asynchronous code allows the program to be executed immediately where the synchronous code will block further execution of the remaining code until it finishes the current one" (Geeks for Geeks, 2020, para. 4).

Some characteristics of async (JavaScript.info, 2021):

- *Async* declares an asynchronous function (async function someName(){...})
- *Async* automatically transforms a regular function into a *promise*
- When called, *async* functions resolve with whatever is returned in their body
- *Async* functions enable the use of *await*

Some characteristics of await (JavaScript.info, 2021):

- *Await* pauses the execution of *async* functions (var result = await someAsyncCall();)
- When placed in front of a *promise* call, *await* forces the rest of the code to wait until that promise finishes and returns a result
- *Await* works only with *promises*; it does not work with *callbacks*
- *Await* can only be used inside *async* functions

Let's look at a *promise*. Here is the syntax:

```
<html>
  <head>
    <script>
      // Declaring a Promise
      let aPromise = new Promise(function (successful, error) {
        // Some code goes in here

        successful();
        error();
      });

      // This is how to execute a Promise
      aPromise.then(
        function (value) {
          // successful code
        },
        function (error) {
          // error handling
        }
      );
    </script>
  </head>
  <body></body>
</html>
```

This is how it can be used:

```
<html>
  <head>
    <script>
      // Declaring a Promise
      let aPromise = new Promise(function (successful, error) {
        var myMoney = 100;

        if (myMoney == 4) {
          successful("Yes.");
        } else {
          error("Error");
        }
      });

      // This is how to execute a Promise
      aPromise.then(
        function (value) {
          logToConsole(value);
        },
        function (error) {
          alert(error);
        }
      );

      function logToConsole(something) {
        console.log(something);
      }
    </script>
  </head>
  <body></body>
</html>
```

There are also instances where async/await is insufficient and we must resort to promises for assistance (JavaScript.info, 2021). One example is when we need to make many distinct asynchronous calls and wait for them all to complete (2021). If we try to perform this with async and await, we will get the following results:

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      async function getABC() {
        let A = await getValueA(); // takes 3 seconds
        let B = await getValueB(); // takes 5 seconds
        let C = await getValueC(); // takes 4 seconds

        return A * B * C;
      }
    </script>
  </head>
  <body></body>
</html>
```

As described by Javascript.info (2021):

Each await call will wait for the preceding one to produce a result before proceeding. Because we are only making one call at a time, the full function will take 12 seconds to complete (3+5+4). This is not the best approach since the three variables A, B, and C are not interdependent. To put it another way, we don't need to know the value of A before we can acquire B. We can obtain them both at the same time, saving us a few seconds of waiting.

A promise.all() is necessary to send all requests at the same time. This ensures that we still obtain all of the results before proceeding, but the asynchronous calls will fire in simultaneously, rather than one after the other.

```
<html>
  <head>
    <script>
      async function getABC() {
        let results = await Promise.all([getValueA, getValueB,
getValueC]);

        return results.reduce((total, value) => total * value);
      }
    </script>
  </head>
  <body></body>
</html>
```

The function will take considerably less time this way. The getValueA and getValueC calls will have completed by the time getValueB finishes. We will essentially decrease the execution to the time of the slowest request rather than a total of the timings (getValueB - 5 seconds).

In-Class Exercise 2

1. How long does it take to execute *doSomething*?

```
<!DOCTYPE html>
<html>
  <head>
    <script>
      async function doSomething() {
        let A = await dance(); // takes 11 seconds
        let B = await danceAgain(); // takes 44 seconds
        let C = await work(); // takes 5 seconds

        return A * B * C;
      }
    </script>
  </head>
  <body></body>
</html>
```

2. How long does it take to execute *doSomething*? (Refer to the above question for timings.)

```
<html>
  <head>
    <script>
      async function doSomething() {
        let results = await Promise.all([dance, danceAgain, work]);

        return results.reduce((total, value) => total * value);
      }
    </script>
  </head>
  <body></body>
</html>
```

MODULE 5 – JAVASCRIPT MINI PROJECTS

Overview

In this module, you will gain experience at writing JavaScript programs utilizing concepts from the previous modules to solve simple, real-world problems.

You will work through two, larger exercises in class, together with your instructor. In each of these exercises you will work together to understand the syntax of, replicate, and run the below code in the coding program that you are using for this course.

These exercises are not graded; they are a hands-on method of learning how to code.

LEARNING OUTCOMES

After completion of this module, you will be able to:

1. **Showcase the JavaScript skills learned in the previous modules to code small projects**

In-Class Exercise 1: JavaScript Trip Calculator

The below code is adapted from Geeks for Geeks (2021):

```
<html>
  <head>
    <script>
      function calculateTheTipAmount() {
        var billAmount = document.getElementById("billAmount").value;
        var serviceQuality =
document.getElementById("serviceQuality").value;
        var numberOfPeopleOnTheBill = document.getElementById(
          "numberOfPeopleOnTheBill"
        ).value;

        if (billAmount === "" || serviceQuality == 0) {
          alert("Please enter values");
          return;
        }

        if (numberOfPeopleOnTheBill === "" || numberOfPeopleOnTheBill <=
1) {
          numberOfPeopleOnTheBill = 1;
          document.getElementById("each").style.display = "none";
        } else {
          document.getElementById("each").style.display = "block";
        }

        var total = (billAmount * serviceQuality) /
numberOfPeopleOnTheBill;

        total = Math.round(total * 100) / 100;

        total = total.toFixed(2);
```

```
        document.getElementById("totalTip").style.display = "block";
        document.getElementById("tip").innerHTML = total;
      }

      document.getElementById("totalTip").style.display = "none";
      document.getElementById("each").style.display = "none";

      document.getElementById("calculate").onclick = function () {
        calculateTheTipAmount();
      };
    </script>
  </head>
  <body>
    <div id="container">
      <h1>Tip Calculator</h1>
      <div id="calculator">
        <form>
          <p>How much was your bill?</p>
          <p>
            $ <input id="billAmount" type="text" placeholder="Bill Amount"
/>
          </p>

          <p>How was this service?</p>
          <p>
            <select id="serviceQuality">
              <option disabled selected value="0">
                -- Choose an Option --
              </option>
              <option value="0.2">Good</option>
              <option value="0.15">It was OK</option>
              <option value="0.05">Terrible</option>
            </select>
          </p>
        </form>

        <p>Number of people sharing the bill?</p>
        <input id="numberOfPeopleOnTheBill" type="text" placeholder="1" />
        people
        <button type="button" id="calculate">Calculate Tip</button>
      </div>
      <div id="totalTip">
        <sup>$</sup><span id="tip">0.00</span>
        <small id="each">each</small>
      </div>
    </div>
  </body>
</html>
```

In-Class Exercise 2: JavaScript Shopping List

The below code is adapted from Sewell (n.d.):

```
<html>
  <head>
    <script>
      const toggleButton = document.querySelector("#toggleList");
      const listDiv = document.querySelector(".list");

      const userInput = document.querySelector(".userInput");
      const button = document.querySelector("button.description");
      const p = document.querySelector("p.description");
      let listItem = document.querySelectorAll("li");

      const addItemInput = document.querySelector(".addItemInput");
      const addItemButton =
document.querySelector("button.addItemButton");

      const removeItemButton = document.querySelector(
        "button.removeItemButton"
      );

      const listItems = document.getElementsByTagName("li");

      toggleButton.addEventListener("click", () => {
        if (listDiv.style.display == "none") {
          listDiv.style.display = "block";
          toggleButton.textContent = "Hide list";
        } else {
          listDiv.style.display = "none";
          toggleButton.textContent = "Show list";
        }
      });

      button.addEventListener("click", () => {
        lastPickedColor = userInput.value;
        listItem = document.querySelectorAll("li");
        for (let i = 0; i < listItem.length; i++) {
          listItem[i].style.color = lastPickedColor;
        }
        p.innerHTML = "The list colour is: " + userInput.value;
      });

      addItemButton.addEventListener("click", () => {
        let list = document.querySelector("ul");
        let li = document.createElement("li");
        li.textContent = addItemInput.value;
        let appendedItem = list.appendChild(li);
        li.style.color = lastPickedColor;
        for (let i = 0; i < appendedItem.length; i++) {
          appendedItem[i].style.color = lastPickedColor;
        }
        addItemInput.value = "";
```

```
      });

      removeItemButton.addEventListener("click", () => {
        let list = document.querySelector("ul");
        let li = document.querySelector("li:last-child");
        list.removeChild(li);
      });

      listDiv.addEventListener("mouseover", (event) => {
        if (event.target.tagName == "LI") {
          event.target.style.textTransform = "uppercase";
        }
      });

      listDiv.addEventListener("mouseout", (event) => {
        if (event.target.tagName == "LI") {
          event.target.style.textTransform = "lowercase";
        }
      });
    </script>
  </head>
  <body>
    <h1 id="myHeading"> Shopping Cart </h1>
    <p class="intro">Your <span class="intro__span">personal</span>
shopping cart.</p>
    <button id="toggleList" class="togglebutton">Hide cart</button>
    <div class="list">
        <p class="description">Type a <span>colour<span> to change cart
colour</p>
        <input type="text" class="userInput">
        <button class="description"> Change List color</button>
        <ul>
          <li>Donair</li>
          <li>Water</li>
          <li>Chips</li>
          <li>Soupr</li>
        </ul>
        <input type="text" class="addItemInput">
        <button class="addItemButton"> Add Item </button>
      <button class="removeItemButton"> Remove Last item </button>
     </div>
  </body>
</html
```

IN-CLASS EXERCISE ANSWERS

MODULE 1

In-Class Exercise 1

1. JavaScript is a client-side programming language with a lot of capability. JavaScript is mostly used to improve the user's engagement with a web page. In other words, you may use JavaScript to make your website more vibrant and engaging. JavaScript is also commonly utilised in the production of games and mobile applications.

2. Java is a complete programming language. In contrast, JavaScript is a coded program that can be introduced to HTML pages. These two languages are not at all interdependent and are designed for different intent. Java is an object-oriented programming (OOPS) or structured programming language like C++ or C, whereas JavaScript is a client-side scripting language.

In-Class Exercise 2

1.
```
<!DOCTYPE html>
<html>
  <body>
    <p> This was last modified <span id="span1"></span> </p>

    <script>
      document.getElementById("span1").innerHTML =
document.lastModified;
    </script>
  </body>
</html>
```

2.
```
<!DOCTYPE html>
<html>
  <body>
    <span id="span1"> <img src="we_are_closed.gif" /> </span>

    <script>
      document.getElementById("span1").innerHTML = '<img
src="we_are_open.gif">';
    </script>
  </body>
</html>
```

3. Answer can be found here: https://www.w3schools.com/js/js_htmldom_methods.asp

MODULE 2

In-Class Exercise 1

1.
```
<html>
  <head>
    <script>
      let candy = "TWIX";
      alert(candy);
    </script>
  </head>
  <body></body>
</html>
```

2.
```
<html>
  <head>
    <script>
      let y = "55";
      alert(y);
    </script>
  </head>
  <body></body>
</html>
```

3.
```
<html>
  <head>
    <script>
      let x = 77;
      let y = 99;
      alert(x + y);
    </script>
  </head>
  <body></body>
</html>
```

In-Class Exercise 2

1. c
2. a, b, d
3. a, b
4. c
5. a

In-Class Exercise 3

1.

```
<html>
  <head>
    <script>
      function addXandY(x, y) {
        return x + y;
      }

      function addXandYandZ(x, y, z) {
        return addXandY(x, y) + z;
      }

      // And the result is
      console.log(addXandYandZ(4, 5, 6));
    </script>
  </head>
  <body></body>
</html>
```

MODULE 3

In-Class Exercise 1

1.

```
<html>
  <head>
    <script>
      var brands = ["Apple", "Samsung", "Mercedes", "Burger King"];

      let brnadsCount = () => {
        return brands.length;
      };

      console.log("Brands Count = " + brnadsCount);

      let isMoreThanTenBranch = () => {
        if (alphabet.length <= 10) {
          return false;
        } else {
          return true;
        }
      };

      console.log(
        "I have more than 10 brands. True of False? " +
isMoreThanTenBranch
      );
    </script>
  </head>
```

```
  <body></body>
</html>
```

In-Class Exercise 2

1. Multiply 4 with 6, alerting the result.

```
<html>
  <head>
    <script>
      alert(4 * 6);
    </script>
  </head>
  <body></body>
</html>
```

Alert the remainder of dividing 12 by 5.

```
<html>
  <head>
    <script>
      alert(12 % 5);
    </script>
  </head>
  <body></body>
</html>
```

MODULE 4

In-Class Exercise 1

1. a

In-Class Exercise 2

1. 60 seconds
2. 44 seconds

SOURCES

BBC. (n.d.). *Arrays and lists.* Retrieved from https://www.bbc.co.uk/bitesize/guides/zy9thyc/revision/1

Future Learn. (n.d.). *Using functions.* Retrieved from https://www.futurelearn.com/info/courses/begin-programming/0/steps/2965

Geeks for Geeks. (2021). *Design a tip calculator using HTML, CSS and JavaScript*. Retrieved from https://www.geeksforgeeks.org/design-a-tip-calculator-using-html-css-and-javascript/

Geeks for Geeks. (2020). *Synchronous and asynchronous in JavaScript.* Retrieved from https://www.geeksforgeeks.org/synchronous-and-asynchronous-in-javascript/

Hartman, J. (2021). *What is JavaScript? Complete introduction with hello world! example.* Retrieved from https://www.guru99.com/introduction-to-javascript.html

JavaScript.info. (2020). *Async/await.* Retrieved from https://javascript.info/async-await

MDN Web Docs. (n.d.-c). *Functions.* Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions

MDN Web Docs. (n.d.-b). *Number.* Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

MDN Web Docs. (n.d.-d). *What is an array?* Retrieved from https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects

MDN Web Docs. (n.d.-a). *What is JavaScript?* Retrieved from https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

Sewell, O. (n.d.). *Shopping list code*. Retrieved from https://codepen.io/o-sewell/pen/mOBjvQ

Tech Target. (2005b). *Loop*. Retrieved from https://whatis.techtarget.com/definition/loop

Tech Target. (2005a). *Object.* Retrieved from https://searchapparchitecture.techtarget.com/definition/object

Tutorial Republic. (n.d.). *JavaScript data types.* Retrieved from https://www.tutorialrepublic.com/javascript-tutorial/javascript-data-types.php

Tutorials Teacher. (n.d.). *JavaScript operators.* Retrieved from https://www.tutorialsteacher.com/javascript/javascript-operators

W3 Schools. (n.d.-a). *JavaScript HTML DOM*. Retrieved from https://www.w3schools.com/js/js_htmldom.asp

W3 Schools. (n.d.-b). *JavaScript string reference.* Retrieved from https://www.w3schools.com/jsref/jsref_obj_string.asp

JavaTpoint. (n.d.). *JavaScript variable.* Retrieved from https://www.javatpoint.com/javascript-variable