**MIT xPRO Data Engineering Certificate**

**JavaScript**

JavaScript is a powerful, yet lightweight, interpreted scripting language that is most frequently used for *client*-side web development. *Client*-side web development allows for a web page to respond to user inputs, such as active mouse clicks, page navigation, and input of information (lakshita 2021). Before JavaScript, web pages were mostly static. Now, JavaScript is used on nearly every web page to generate a dynamic user experience.

JavaScript is powerful because it supplies objects to control the browser and its DOM. Unlike most programming languages, JavaScript has no concept of input or output. It is designed to run as a scripting language in a host environment, and the host environment provides mechanisms for communication with the outside world. The most common host environment is the browser.

**Characteristics of JavaScript**

| Dynamically typed | This language can receive different data types over time. |
| --- | --- |
| Case-sensitive | JavaScript is case sensitive. Ensure that you always use consistent capitalization for all *functions*, keywords, and other identifiers. |
| Lightweight | JavaScript is considered a lightweight programming language because variables’ types don’t need to be declared, and it’s easy to use compared to other languages, such as Java or C++. |
| Designed for event handling | Handling events is the main feature of JavaScript. It can quickly respond on the website when the user tries to perform any operation. |
| Interpreter-centered | JavaScript is built as an interpreter-centered language. This allows the interpreter to read the code and run it to produce the user output without the use of the compiler (lakshita 2021). |

**How to Implement JavaScript**

JavaScript is implemented along with the standard HTML file. JavaScript can be a standalone file with the extension .js. Think of JavaScript as a layer to your HTML page. It can be included directly in the HTML <script> *tag*. However, it is more common to link an external source JavaScript file. It is also common to have multiple JavaScript *tags* within an HTML file.

Next, you will practice including JavaScript in HTML using an example from [Project 23.1](https://classroom.emeritus.org/courses/7145/assignments/187436).

Below are the steps for creating the mitcourses\_graph.html file, which is used in Project 23.1.

| <!DOCTYPE html>  <html>  <script src="d3.tip.js"></script>  <script src="words.js"></script>  <script src="mitcourses\_graph.js"></script>  </html> |
| --- |

This code will link three JavaScript files: d3.tip.js, words.js, and mitcourses\_graph.js.

Next, assign the target id *attribute*, so the HTML <body> *tag* can be referenced uniquely. Then, call the JavaScript graph()*method* for data to be processed and displayed to the browser.

| <!DOCTYPE html>  <html>  <script src="http://d3js.org/d3.v3.min.js"></script>  <script src="d3.tip.js"></script>  <script src="words.js"></script>  <script src="mitcourses\_graph.js"></script>  <body>  <div id="target"></div>  </body>  <script>  graph();  </script>  </html> |
| --- |

Note that the following line from the code above will link the D3 *library* to your HTML page: <script src="http://d3js.org/d3.v3.min.js"></script>.

**JavaScript Syntax**

In JavaScript, *statements* end with semicolons (sheyakhare1999 2019). Below is an example of valid JavaScript code:

| <!DOCTYPE html>  <html>  <body>  <h2>Welcome</h2>  <p id="Emeritus"></p>  <script>  var a, b, c;  a = 2;  b = 3;  c = a + b;  document.getElementById(  "Emeritus").innerHTML =  "The value of c is " + c + ".";  </script>  </body>  </html> |
| --- |

In JavaScript, keywords are reserved words that have a specific meaning when used in a code *block*. Thus, they cannot be used as random variables because they communicate that a particular operation will be performed (sheyakhare1999 2019). Below is a list of reserved JavaScript keywords:

**Reserved JavaScript Keywords**

| break | This keyword is used to “terminate a *loop* or *switch''* (sheyakhare1999 2019). In JavaScript, a *switch* is a construct that evaluates expressions based on certain values. If there is a match, the associated *block* of code is executed; otherwise, the code skips the *statements* within the *switch* construct. |
| --- | --- |
| continue | This keyword is used to “skip a particular iteration in a *loop* and move to the next iteration” (sheyakhare1999 2019). |
| do… while | This JavaScript construct is similar to a Python while *loop*. In this construct, the *statements* written within the do *block* are executed as long as the condition in the while *block* is true (sheyakhare1999 2019). |
| for | This JavaScript construct is the same as a Python for *loop*. The for *loop* executes a *block* of *statements* as long as the condition remains true (sheyakhare1999 2019). The code exits the for *loop* when the condition is no longer true. |
| function | This keyword is used to “declare a *function*” (sheyakhare1999 2019). It is equivalent to the Python def keyword, which is used to declare a user-defined *function*. |
| return | This keyword is used to exit a *function* and to return a value, if specified. It is equivalent to the Python return keyword, which is used at the end of a *function* to return a variable. |
| var | This keyword is used to “declare a variable” (sheyakhare1999 2019). |
| switch | In JavaScript, this keyword is used to define a construct that evaluates expressions based on certain values. |

**Global vs. Local Variables**

JavaScript is a dynamically typed programming language; this means that when declaring variables, it is not necessary to include their data type. Furthermore, in JavaScript global, as in Python, variables can be declared outside of a *function’s* scope and used throughout the script.

The code example below demonstrates how global and local variables work in JavaScript. The Name variable is declared outside of theMyFunction()*function*. In JavaScript, Name is a global variable in the sense that its assigned value can be used throughout the script. On the other hand, the num variable is declared inside the MyFunction()*function*. In this case, num is a local variable defined within the *function*, meaning that it can only be used inside the *function* declaration. After it is called, theMyFunction()*function prints* the values assigned to both the Name and num variables.

| <script>  // Global variable declaration  var Name="Apple";  // Function definition  function MyFunction() {    // Local variable declaration  var num = 45;    // Display the value of Global variable  document.writeln(Name);    // Display the value of local variable  document.writeln("<br>" + num );  }  // Function call  MyFunction();  </script> |
| --- |

JavaScript is debatably the most commonly used programming language in the world, and interest in it has only been increasing. Once you understand the fundamentals of JavaScript, you will be able to apply that knowledge to your hands-on work in the data engineering industry.

**Data-Driven Documents (D3) Library**

**What Is the D3 *Library*?**

The D3 *library* is a JavaScript *library* used for visualization. The goal of D3 is to associate the data that you want to visualize with the DOM. This allows you to directly manipulate, change, or add to the DOM, thus enabling greater control over how the web page will appear.

D3 can be used in many ways. An extensive list can be found on the [D3 Gallery](https://observablehq.com/@d3/gallery) page. In this mini-lesson, you will learn how to create a bubble chart to display word frequencies.

**How to Create a Bubble Chart Using D3**

To follow along with this mini-lesson, download the [d3\_bubble\_chart\_example.html](https://classroom.emeritus.org/courses/7145/files/1904792/download) file, which contains the code to create a bubble chart to display word frequency using the D3 *library*.

After downloading the file, open it using VS Code. The steps below will guide you through the code in the index.html file.

The code below demonstrates how to import the JavaScript D3 *library* into your HTML file. This will allow you to use the *library* without installing it locally. Next, you will manually define a dataset named children. This dataset is an *array* that contains each word and how many times it occurs in your data. In the example below, only four words are defined: Olivia, Mark, Jessica, and Alex.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<script

type="text/javascript"src="https://d3js.org/d3.v4.min.js">

</script>

<style type="text/css">

</style>

</head>

<body>

<script type="text/javascript">

dataset = {

"children": [{"Name":"Olivia","Count":78},

{"Name":"Mark","Count":35},

{"Name":"Jessica","Count":30},

{"Name":"Alex","Count":55}]

};

Next, define the bubbles for the diagram on your HTML page. You can do this by calling the append()*function* and by passing circles as *arguments*. This is demonstrated in the code below:

node.append("circle")

.attr("r", function(d) {return d.r;})

.style("fill", function(d,i) {return color(i);});

Next, you can add the information about your words to your HTML code to display each word and its frequency. The code below uses the append()*function* with text as an *argument* to enhance your bubble diagram by adding labels to the circles. This *function* reads the name of every word in your dataset and adjusts the font to scale along with the size of the corresponding bubble.

node.append("text")

.attr("dy", ".2em")

.style("text-anchor", "middle")

.text(function(d) {

return d.data.Name.substring(0, d.r / 3);})

.attr("font-family", "sans-serif")

.attr("font-size", function(d){

return d.r/5;})

.attr("fill", "white");

Finally, the frequency of each word is added by calling the append()*function* with text as an *argument* once again. This time, this *function* enhances your bubble diagram by retrieving the number of times each word appears in the dataset. It then adjusts the font for the frequency count to be at scale with the corresponding bubble.

node.append("text")

.attr("dy", "1.3em")

.style("text-anchor", "middle")

.text(function(d) {

return d.data.Count;})

.attr("font-family", "Gill Sans", "Gill Sans MT")

.attr("font-size", function(d){

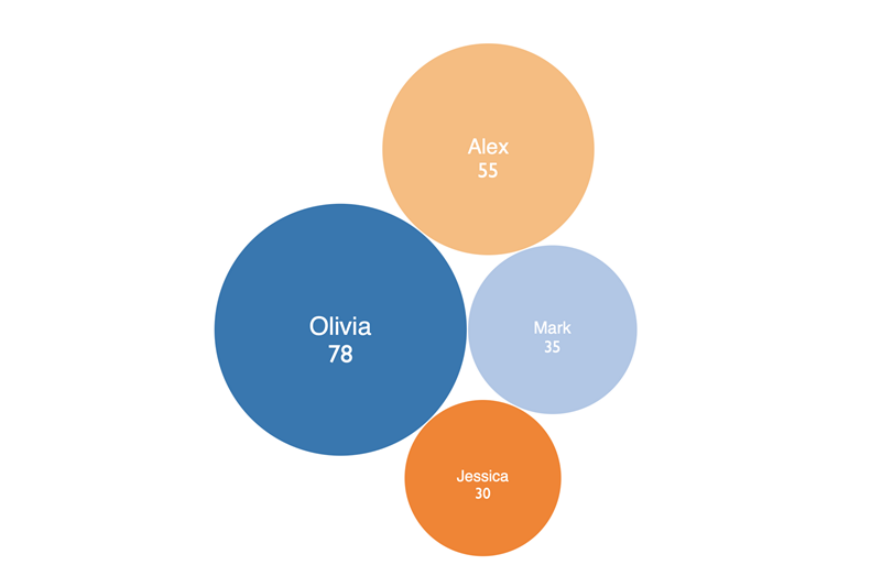
return d.r/5;})

.attr("fill", "white");

To experiment with this code, open a new browser window and navigate to the following URL:

file:///YOUR\_PATH\_HERE/d3\_bubble\_chart\_example.html

You should see the following bubble chart:



Now you know how to create a bubble chart using the D3 *library*.