Lab 1: Introduction to JavaScript and Node.js

1 Purpose of this lab

In this lab, we introduce JavaScript and Node. JavaScript is the default scripting language used in all standard web browsers (e.g. Chrome, Safari, FireFox, even Internet Explorer) and is therefore the language for implementing the majority of web applications. Node brings the power of JavaScript to the server-side of an application. Consequently, JavaScript can be used on both the client-side and server-side of a web application.

2 Overview to JavaScript

Taken from: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction

JavaScript is a cross-platform, object-oriented scripting language. It is a small and lightweight language. Inside a host environment (for example, a web browser), JavaScript can be connected to the objects of its environment to provide programmatic control over them.

JavaScript contains a standard library of objects, such as Array, Date, and Math, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects; for example:

- Client-side JavaScript extends the core language by supplying objects to control a browser and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on an HTML form and respond to user events such as mouse clicks, form input, and page navigation.
- Server-side JavaScript extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a database, provide continuity of information from one invocation to another of the application, or perform file manipulations on a server. Node is the primary server-side Javascript implementation used today.

Throughout this course we will be using JavaScript for the majority of the exercises. Specifically, we will be using the ES2015 standard. If you are new to JavaScript and/or the ES2015 standard then you should take some time now to familiarise yourself. There is an abundance of information available online.

Below are some useful resources:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript

- https://leanpub.com/understandinges6
- https://developers.google.com/web/fundamentals/getting-started/
- https://leanpub.com/understandinges6
- http://exploringjs.com/
- https://leanpub.com/es6-in-practice
- http://javascript.crockford.com/
- https://github.com/ericdouglas/ES6-Learning#articles--tutorials

3 What is Node.js?

From their website (nodejs.org):

“Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.”

But what does this mean? Let’s break it down:

“Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.”

V8 compiles and executes JavaScript source code, handles memory allocation for objects, and garbage collects objects it no longer needs. JavaScript is most commonly used for client-side scripting in a browser, being used to manipulate Document Object Model (DOM) objects for example. The DOM is not, however, typically provided by the JavaScript engine but instead by a browser. So V8, as a pure Javascript engine, does not include any methods for DOM manipulation. V8 does however provide all the data types, operators, objects and functions specified in the ECMA standard. http://node.green/ shows exactly which ES2015 features are supported by each Node version. It shows that version 6.4.0 and above have the best support.

“Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.”

Event driven: “In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads.” - https://en.wikipedia.org/wiki/Event-driven_programming

Non-blocking: Read a short blog post on the difference between blocking and non-blocking code at:
https://medium.com/@hengkiardo/blocking-versus-non-blocking-code-d3bde835062f

“Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.”

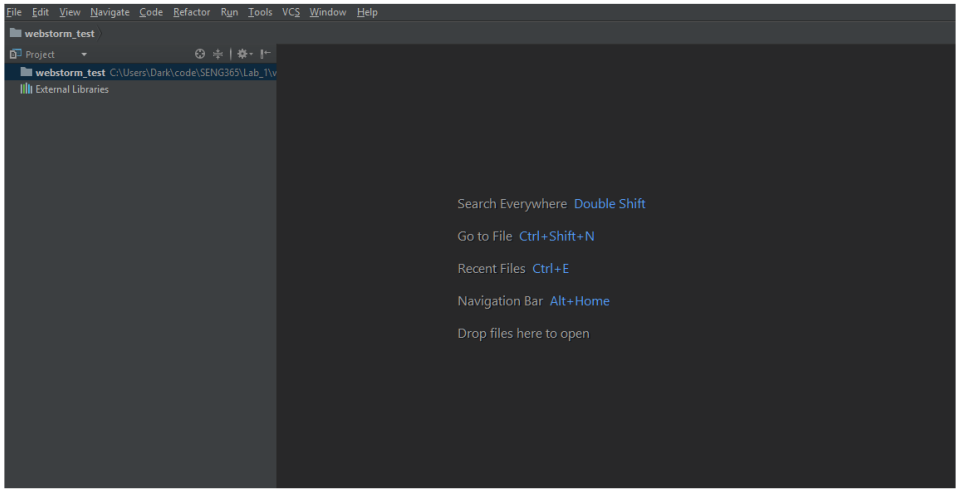
NPM: “npm , short for Node Package Manager, is two things: first and foremost, it is an online repository for the publishing of open-source Node projects; second, it is a command-line utility for interacting with said repository that aids in package installation, version management, and dependency management.” - https://docs.nodejitsu.com/articles/getting-started/npm/what-is-npm/

3.1 Before we start: Running scripts in WebStorm

Many of the seasoned JavaScript developers amongst you may already have a preferred workflow for writing web applications. For those of you who do not, then we recommend using the WebStorm IDE. Webstorm is simple to use and already installed in the labs. For any of you wishing to use Webstorm on your personal machines, then you get a free version using your university email account (you may have to reconfirm that you are a student each year).

Running scripts on Webstorm is easy:

1. Open Webstorm
2. Select ‘create new project’
3. Create an empty project (make sure to rename your project first and provide a project path). The IDE will open your project directory as shown below.



4. Right click on your project directory and select: New > JavaScript File
5. When prompted for a filename, call it ‘test.js’
6. Your new file will open, copy and paste the below code into it:

```
console.log("Hello!");
```

7. Save your file (CTRL + s) and run your script by clicking Run > Run test.js or using the shortcut SHIFT + F10. **Alternatively:** you can run your scripts by opening your terminal (View > Tool Windows > Terminal) and entering the command ‘node test.js’

Read more: https://www.jetbrains.com/help/webstorm/quick-start-guide.html

3.2 Getting Started: Download and installation (if using a personal computer)

1. Download latest version of Node from http://nodejs.org/download.
2. Run the downloaded file and navigate through the wizard.
3. Confirm the installation by opening your terminal and typing 'node.' You should be presented with the node prompt ('>').
4. Confirm that node has installed correctly by moving onto Exercise 1.

3.3 Exercise 1: Hello World

1. Create a file named **exercise1.js** on your machine
2. Open the file and insert the following code

```
/* Hello, World! program in node.js */
console.log("Hello, World!")
```

3. Open your terminal, navigate to the directory where your file was created and type: **node exercise1.js**
4. If node was successfully installed, 'Hello, World!' should be outputted to the console.

3.4 Exercise 2: Write a Web Server

In Node, we load modules into other modules using the 'require' directive. (c.f 'import' in Python).

1. Create a new file called **exercise2.js**
2. Use the 'require' directive to pull in the http module into a variable for later use.

```
const http = require("http");
```

3. Now use the 'http' module's 'createServer' function to create a new server. The example code below ignores the content of any request given and instead will just return a 200 message with the text 'Hello World.'

```
http.createServer(function (request, response) {

   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});

   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);
```

```
console.log('Server running at http://127.0.0.1:8081/');
```

The documentation for the createServer function can be found at: https://nodejs.org/api/http.html#http_http_createserver_requestlistener

4. Open your terminal and run: **node exercise2.js**
5. Open your browser and navigate to http://127.0.0.1:8081 or http://localhost:8081

3.5 Exercise 3: Handling URL parameters

Next we want to be able to retrieve URL parameters so that we can use them in our applications. Read https://en.wikipedia.org/wiki/Query_string for an explanation of what URL parameters are.

1. Make a copy of your exercise3.js file and call it exercise3.js
2. Use the require directive to pull in the 'url' module. The 'url' module allows us to easily parse URLs.

```
const http = require("http"), URL = require('url').URL;
```

3. Next, edit the contents of your createServer function. We want to first add a listener on the end event of the HTTP request. This event will parse the URL for the parameters and then return a 200 HTTP response with the data in the response body.

```
http.createServer(function (request, response) {
        const parameters = new URL(request.url,
'http://localhost').searchParams.toString();
        // write the response
        response.writeHead(200, {
                'Content-Type': 'text/plain'
        });

        response.end('Here is your data: ' + parameters);
}).listen(8081);
```

4. Again, run your server and then go to your browser and navigate to the server's URL. Add parameters to your URL, they should be listed in your browser (e.g. http://localhost:8081?name=Jake&age=35).

3.6 Exercise 4: Putting it all into practice

In this exercise, we will create a server that contains a shopping list. When a user navigates to the server with the parameter 'item_num' set, the server will respond with the name of the item at the specified index of 'item_num.'

1. Create a new file called exercise4.js
2. Import the 'http' and 'url' modules using the require directive
3. Create a server
4. Inside the server, create a variable that contains a list of items that are typical of a shopping basket (e.g. milk, bread, eggs, flour).

5. Attach a listener on the ‘end’ event of the HTTP request
6. Parse the URL for the parameters
7. Find the value of ‘item_num,’ you may need to do some research here on how to find a specific parameter.
8. Return a 200 HTTP response that writes out to the browser the item in the list at position ‘item_num.’

That’s it for lab 1, you should now have a basic knowledge of both JavaScript and Node. In the next lab, we will be creating our first API in Node.