1 Objectives

In this reading material, the objectives are:

1. To give background information on Node.js and the ingredients required to compile and run code using it
2. To describe how to install Node.js
3. To describe the use of the command-line interface to compile and run code
4. To describe the use of Node.js to identify errors in code
5. To describe and use an implementation of vectors and dynamic arrays using JavaScript Arrays

2 Background

2.1 What is Node.js?

JavaScript is a programming language used in web browsers, but can we run JavaScript code outside of a browser environment? The answer is yes, and Node.js is a solution to this. Once installed on your device, Node.js allows you to run your JavaScript code in an environment free from HTML, CSS and so on. So you can develop your own code and programs without resorting to a web browser, or even without an internet connection.

There is a lot more to Node.js than this, but during this course we will just use its ability to compile and run JavaScript code. Once Node.js is installed, to write code you only need a text editor, or ideally an integrated development environment (IDE). To compile and run the code we will use the command-line interface (or console user interface) of your OS.

2.2 Command-line Interface (CLI)

It is common to interact with their machine solely through the graphical user interface (GUI). If you ever wanted to navigate through the folders and files on your machine without ever using your mouse, then you can use the command-line interface (CLI): you can replicate the functioning of the GUI just by entering instructions through lines of text one at a time. Depending on your OS, your CLI will be implemented by particular programs:

- For MacOS, it is called Terminal: go to Spotlight Search, enter Terminal for quickest way to launch it
- For Windows, it is called Command Prompt: type `CMD` into your Windows search box and click on `Command Prompt` when it appears
- For Linux, it is generically the command line, and will typically be called Terminal

We will refer to these programs as the CLI for convenience. For our purposes, we will use the CLI for the following:

- To run JavaScript code
- To print the results of calling `console.log()`
- To identify bugs in the code

- To install Node.js, if there is no available installer.

In the CLI your application will be inside one particular directory or folder at any one time, just as your Documents folder in the GUI, for example, is showing everything inside that folder and has the address of that folder. We can use simple commands in the CLI to move between folders and list what is inside them. There are three important commands in the CLI for these purposes:

- to change to a particular directory (folder): `cd /d folder` for Command Prompt, and `cd folder` for Terminal
- to list files and sub-folders in a directory (folder): `ls` for Terminal and `dir` for Command Prompt
- to find out which directory (folder) you are in: `pwd` for Terminal and `cd` alone for Command Prompt

2.3 Writing code

To write our JavaScript code, all we need is a method for writing text. For MacOS, this could be TextEdit; for Windows, you could use Notepad (or Word). Your code files need the format suffix ".js"; a ".txt" or ".docx" file, for example, will not be processed by Node.js as a piece of JavaScript code. If you save your code as a text file (e.g. with the suffix ".txt"), you could just rename the file by replacing the suffix with ".js". Also, be careful about file formatting. Make sure you have a Plain Text format.

One problem with text editors is that they do not inherently know that you are writing code. Therefore, if you make a small mistake, you might not catch it in the text editor, even if you know your code is not working properly. This is why it is highly recommended that you use integrated development environments (IDEs). Within an IDE, it is much easier to organise, edit and correct code, since the IDE "knows" that you are trying to code.

You should have had experience with Brackets, which works within a web environment, but in which you can write and save JavaScript code (within a project). Two nice examples of IDEs are Sublime Text (sublimetext.com). Technically, the full version of Sublime Text is not free, but you can use it on a "trial" version indefinitely without paying. Close (and then ignore) any warnings asking you to pay. Note that in the video reviewing loops in JavaScript, I quickly review these IDEs.

3 Getting started

3.1 Installing Node.js

For the following, you should ideally use one of the following browsers: Firefox, Chrome, Safari. Internet Explorer should also work. Go to nodejs.org and then "Downloads" and follow the instructions to download Node.js. Once on the "Downloads" page, select the "LTS: Recommended for Most Users" tab. For Windows and MacOS, there is an Installer which will do all the work for you. For other platforms such as Linux, you might need to use the CLI to install Node.js. At the bottom of the "Downloads" page, you can find multiple options for installing Node.js (through a package manager, or otherwise).

3.2 Setting up a folder

Once you have installed Node.js, you should create a folder in "Documents": name the folder `hello` (or something suitably memorable)[1]. Once you have created the folder, you can now access it using your CLI. To do this, do the following:

1. Open your CLI (Terminal or Command Prompt) and do the following:

[1] Warning: it becomes very fiddly to navigate through folders with spaces in the name. There is a solution, but it's best to avoid spaces in folder names

- For MacOS (Linux works similarly), when you launch Terminal you should already be in the folder `/Users/`*`UserName`* where *UserName* is your user name. If you are not sure enter `pwd` and if you are not in this folder, enter:

  ```
  cd Users/UserName
  ```

  and now enter the following to go to your new folder:

  ```
  cd Documents/hello
  ```

  Next to see if there is anything in the folder, enter

  ```
  ls
  ```

- For Windows, your directory path will be something like `\C:\Users\`*`UserName`*`\Documents\pscs1` where *UserName* is your user name. In Command Prompt, enter

  ```
  cd /d C:\Users\UserName\Documents\hello
  ```

  and you will now be in `pscs1`. Once you're in, enter

  ```
  dir
  ```

  if there is nothing in the folder, the information returned in the window will indicate this.

Leave your CLI window open as we will return to it.

2. Open your IDE, create a new blank document, and write the following (do not copy and paste as it might introduce formatting errors):

   ```
   console.log("Hello, World!")
   ```

   Save the file as `hello.js` and make sure it is saved to the folder `hello`.

3. Return to your CLI and enter the relevant command to list all the files in the directory. Check that `hello.js` is there. If it is not, then move it to this directory (you can use the GUI for this). Leave your CLI open.

3.3 `Hello, World!`

After doing all of the above, you are now ready to use Node.js to compile and run your JavaScript code saved as `hello.js`. You should still be in your folder `hello`, so now enter the following into your CLI (and press enter):

```
node hello.js
```

If everything went well, below this line you should see printed `Hello, World!`. Node.js compiled and executed your JavaScript code. The command `console.log()` in the code executes the printing of something (inside the brackets) to the console, which in our case is the CLI. We will use CLI and console interchangeably because of this.

If it didn't go well, you should have got an error message in the console – this is Node.js returning an error message. A bug might be any formatting imposed by your text editor, e.g. TextEdit likes to use "SmartQuotes" leading to symbols, which cannot be interpreted by Node.js.

At this point, it's good to point out that this is the general method for executing JavaScript code in Node.js through the console: `node yourfile.js`. Or simply `node yourfile`.

3.4 Errors

Node.js can return error messages in the CLI to inform us what went wrong. Sometimes it requires some effort to see what went wrong, but it will at least indicate where the error is.

Go back to `hello.js`. We will now modify it. Replace `console.log("Hello, World!")` with

```
console.log(Hello, World!)
```

That is, removing the quotation marks from the previous version. When you run this code you should an error like the following:

```
/Users/mattyhoban/Desktop/hello.js:1
(function (exports, require, module, __filename, __dirname) { console.log(Hello, World!)
                                                                      ^^^^^

SyntaxError: missing ) after argument list
    at new Script (vm.js:79:7)
    at createScript (vm.js:251:10)
    at Object.runInThisContext (vm.js:303:10)
    at Module._compile (internal/modules/cjs/loader.js:656:28)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:699:10)
    at Module.load (internal/modules/cjs/loader.js:598:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:537:12)
    at Function.Module._load (internal/modules/cjs/loader.js:529:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:741:12)
    at startup (internal/bootstrap/node.js:285:19)
```

We can see here that the response in the CLI is telling us something is wrong with `World!` using the `^^^^^` characters. It is saying `SyntaxError` so there is something wrong with the syntax and suggests that we are missing a bracket after `World`. That is, the compiler does not like the fact that we have `!` after `World`. Let's fix this: replace `console.log(Hello, World!)` with

```
console.log(Hello, World)
```

Now enter `node hello` in your CLI. You should get the following error:

```
(function (exports, require, module, __filename, __dirname) { console.log(Hello, World);
                                                                          ^

ReferenceError: Hello is not defined
    at Object.<anonymous> (/Users/mattyhoban/Desktop/hello.js:1:75)
    at Module._compile (internal/modules/cjs/loader.js:688:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:699:10)
    at Module.load (internal/modules/cjs/loader.js:598:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:537:12)
    at Function.Module._load (internal/modules/cjs/loader.js:529:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:741:12)
    at startup (internal/bootstrap/node.js:285:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:739:3)
```

Now we do not have a `SyntaxError`, but a `ReferenceError`. This is another common error when we refer to a variable that is not defined in the code. Here we see that the compiler thinks that `Hello` is a variable that should be defined, but is not.

The real problem is that `Hello, World!` should be stored as a string as in the original code, but there are no quotations. The computer didn't know we wanted to print a string, so it can only give limited information. A lot of programming is problem solving on the basis of limited information!

4 Vectors and dynamic arrays as JavaScript arrays

Throughout the course so far we have focused on abstract and concrete data structures independently of any programming language. In the rest of the course we will implement algorithms in JavaScript as well as studying them abstractly.

JavaScript works at a high level, so we do not need to describe each and every operation done by the computer; a single line of code may involve multiple operations done by the computer. JavaScript also has an abstract data structure built in to the language, in particular it has the dynamic array in its syntax: these are JavaScript arrays. Do not confuse JavaScript arrays with the array data structure. To reiterate, a JavaScript array is a dynamic array, and its implementation is hidden from the programmer. Because a JavaScript array is a dynamic array, it can also be used to implement a vector. When we program in JavaScript, vectors will be implemented with JavaScript arrays.

A JavaScript array is an object: it is a collection of data where the data can be accessed in particular ways. There is also a special syntax for creating arrays. An empty array called `arr` can be created in the following way:

```
var arr = [];
```

If we wish to specify elements of a JavaScript array, we could do something like this following:

```
var arr = [1, 2, 3];
```

which will store the numbers 1, 2 and 3 in three elements.

Now we can go through each of the operations of a dynamic array and show how they are implemented with JavaScript syntax.

- length: for an array called `arr`, its length will be returned by `arr.length`. For example, we can declare a new variable with its value being the length of the array:

```
var l = arr.length;
```

- select[k]: to obtain the value stored in an element of an array, we use square brackets after the name of the array with element number inside the brackets. Note that with JavaScript arrays the index of the first element is 0, not 1. So to find the first element of the array and assign it to a variable we can do the following:

```
var l = arr[0];
```

  To find the last element of the array we do the following:

```
var l = arr[arr.length - 1];
```

- store![o,k]: to write a value to an element we just use the equality as an assignment, so if we wished to write the value 3 to the second element of array `arr` then we write:

```
arr[1] = 3;
```

- removeAt![k]: there are multiple ways to remove elements from a JavaScript array. One way is to use the method called `splice()`. For example, to remove the third element of the array `arr` we use:

```
arr.splice(2,1);
```

  The first term in the brackets says where the element will be remove, and the second term indicates the number of elements to be removed.

- insertAt![o,k]: as with the last operation, there are multiple ways of adding an element to a JavaScript array at index k with value o. Just as with the last operation, we can use `splice()` to add elements. For example, to add a new element at the second element with the value 10 we write:

```
arr.splice(1,0,10);
```

  The first term in the brackets says where the element will be added, the second term indicates the number of elements to be removed (which is now 0), and the final term in the brackets specifies the value at the new element.

4.1 Copying data from array to another

It is useful to create new JavaScript arrays from the data in another. For example, we might want a second array to contain manipulated data from the first array but not lose the data from this first array. One problem with JavaScript arrays is that when we try to assign the value of one element of an array to the element of another array, we pass a *reference* to the original data, not the value itself. It is as if you copy a pointer to a piece of data and not the data. To properly copy just the value we use `slice()`. So if we want to copy all the values of an array called `arr` to an array called `arr2` we use:

```
var arr2 = arr.slice();
```

To copy a limited number of values, we can put some values in the bracket. For example, to copy values from the second element to the fifth (inclusive) of `arr` to `arr2` we write:

```
var arr2 = arr.slice(1,5);
```

Let's look at an example of pseudocode and see how we can implement it with JavaScript. Here's the pseudocode:

new DynamicArray d
$d[1] \leftarrow 1$
for $2 \leq i \leq 4$ **do**
    $d[i] \leftarrow i \times (i-1)$
end for
new DynamicArray s
$s[1:2] \leftarrow d[2:3]$
$x \leftarrow s[1] \times s[2]$

Go to your JavaScript file called `hello.js`, remove all existing code and add the following code:

```
var d = [1];
for (var i = 1; i < 4; i++) {
  d[i] = i * (i + 1);
}
var s = [];
s = d.slice(1,3);
var x = s[0] * s[1];
console.log(d);
console.log(s);
console.log(x);
```

Now when you run this program using the CLI using the command `node hello`, you should get the following printed in your CLI:

```
[ 1, 2, 6, 12 ]
[ 2, 6 ]
12
```

We will be using JavaScript arrays to handle vectors in searching and sorting algorithms. However, understanding the abstract data structure of the JavaScript array being an implementation of the dynamic array will allow you to implement other abstract concepts.