Programming in JavaScript

Section 2: JavaScript Outside the Browser & Transpiling

2.1 NodeJS

NodeJS is a standalone JavaScript runtime outside of the browser (behaviourally it is working like JavaScript in the browser). This allows developers to program the front and backend of the web application in one language. NodeJS also introduces the native event driven program flow to backend development. Instead of running a process that waits for a user action, NodeJS will fire an event whenever an action occurs and handlers can be programmed to respond to that action (similar to attaching event listeners to HTML elements in the browser).

While the main thread of NodeJS is single threaded like JavaScript, NodeJS is capable of running processes concurrently without blocking (i.e. non-blocking).

NodeJS can be used for any number of things such as Web Servers, Automated Tests, Build Scripts (Transpiling), etc.

To download node.js head over to the NodeJS website https://nodejs.org/en/. Run the following terminal command to check whether NodeJS is installed on your machine. This should return the version number in the terminal:

```
node -v
```

2.2 Running NodeJS

Unlike scripts in the browser you will not be able to run a NodeJS script by opening up a webpage. The NodeJS script would either need to be written in the terminal, invoked from another script or write a shebang to the beginning of the NodeJS file and make the script executable (the last example shown below).

```
#! usr/local/bin/node
console.log('I Started');
```

We can create a .js file and add our NodeJS code which looks the same as writing JavaScript as seen in the example to the right. To execute the code go to the terminal and write node followed by the path to the script file (you may not need to specify the path if you are already in the directory containing the script within the terminal whereby you can reference the script file name itself).

```
script.js > ...
function showGreeting(text) {
  console.log('Hello' + text);
};

showGreeting('World');
```

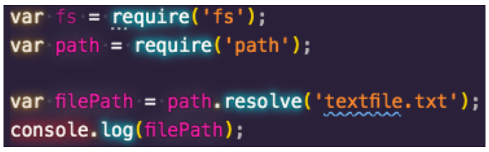
```
$ node script.js
HelloWorld
```

The main difference between JavaScript in the browser and NodeJS is that there are no associated web page, no native DOM and no browser window object. However, there is a global scope (global Object). This global object of NodeJS is similar to what the window object is to JavaScript in the browser i.e. the top level object.

NodeJS global Object does not have document or browser-specific functionality. However, it does have the ability to interact with the file system, spinning up a new process, etc.

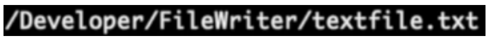
The require function imports global modules, local scripts and JSON files. Global modules include fs and path modules which allows interaction with the File System. The require function also allows breaking up projects into multiple files.

The fs module is an object with functions for interacting with the files system i.e. reading and writing to files on the system. The path module is an object with functions for retrieving and constructing file paths in a system agnostic way.



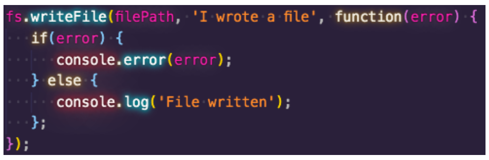
```
var fs = require('fs');
var path = require('path');

var filePath = path.resolve('textfile.txt');
console.log(filePath);
```

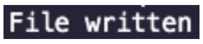


```
/Developer/FileWriter/textfile.txt
```

The .resolve function provides the file path to the file. This will allow us to write a new text file in the file path using the fs module object functions.



```
fs.writeFile(filePath, 'I wrote a file', function(error) {
  if(error) {
    console.error(error);
  } else {
    console.log('File written');
  };
});
```



```
File written
```

textfile.txt

This code will create a new textfile.txt file within the file path with the text of I wrote a file. If executed successfully the terminal will print File written else an error message.

The writeFile is a function from the fs module and takes in some arguments. The first is the file path (either an absolute or a relative path) to the file, the second argument is the string data to write to the file and the third argument is a callback function to call when the file is successfully written or if an error occurs. The callback function is invoked once asynchronously once the file is either written or error occurs. This approach is foundational in NodeJS and what differentiates itself with other languages. This approach should be reminiscent of events in the browser such as the window.onload( ) function.

In NodeJS it is conventional to have a single callback function as the last argument and also within that function it is also conventional to have the first argument being a reference to an error object and then any number of arguments after it depending on the function. If no error occurs the error object will be null.

A NodeJS script will automatically terminate when all asynchronous operations complete.