Innocent Kironji

Node.js is not a formal/ independent programming language. It is not actually a special dialect of JavaScript either. Node.js is more of a programming engine, based off Google Chrome's V8 JavaScript engine, that runs JavaScript. Node.js is a JavaScript runtime environment that processes incoming requests in a special loop known as the event loop (Node.js Foundation). The only notable changes Node.js makes from standard JavaScript is the inclusion of several extra libraries to assist with I/O and networking. As such it is a very powerful tool for using JavaScript outside of a server. "Node.js allows you to use all these innovative JavaScript projects on the server the same as on the client browser," (xxiii, Syed). To be more specific, many tools in Node.js are present in JavaScript in either the standard language or through other external libraries, especially when using JavaScript in conjunction with the V8 engine. As such, in this paper, I will talk both about JavaScript and Node.js but I will mainly focus on the tools of JavaScript that are best adapted to working on a terminal, text editor or client browser. However, Node.js does add enough additional libraries and functionality that is not possible in web browsers to distinguish it from programming JavaScript in any other way.

The most notable features of Node.js are the event loop and many of its other asynchronous operations. Node.js "presents an event loop as a runtime construct instead of as a library," (Node.js Foundation). There is nothing you must do to have Node work in events and run and catch different events at different times. Many of the necessary events—such as keypress, stdin/ user input, file input/ output, server updating—are built in to Node.js upon installation without having to include external libraries. These functions all come in Node's standard library (Node.js Foundation).

Many of the new features Node.js brings with its libraries that cannot be done with JavaScript on web browsers relates to file I/O. Node's main file input and output functions can

be accessed and utilized through the 'File System' module (Node.js Foundation). This module allows the programmer the ability to read files, write to files and append to files—either before or after reading the file. These functions all work the same as they would in Functional and Object-oriented programing languages. For example, using the writing functions in the File System module can allow you to delete the contents of an existing file and overwrite it or, if the file does not exist, create an entirely new file and save that file after writing to it. Functions like these are difficult to replicate in other forms of JavaScript—especially web-based JavaScript—which gives Node.js an edge.

Reflection

Node.js seemed to be built perfectly for some problems and more challenging for others. Of all the problems the easiest to implement using Node.js was problem 4—generating random numbers. This problem heavily relied on being able to handle multiple tasks at the same time. Because of its asynchronous properties Node.js is excellent at this. Node.js can handle multiple tasks concurrently (Node.js Foundation). Also, because many of it's asynchronous functionality is event-based Node is even more suited to a problem like this where a task must be completed until a specific event is recognized; in this case a keypress event. Problem 6—Golomb Sequence—was also an easy problem for Node because Node is extremely fast and efficient in code execution—largely due to its use of Google's V8 engine.

The problems that gave the most trouble were problems that required multiple unique inputs. Node's standard library uses the 'Readline' module for user input (Node.js Foundation). This module is simply an event that waits for keyboard input from the user. As such if the input needed is unique or changes then a new event must be setup to handle this different event. Also, problems that required the same information—such as variables—across different events were

also difficult to implement unless using things like global variables. Events work similarly to functions where a variable goes out of scope after an event stops executing, switches to another event or the event responds differently. For example, the event for reading user input can be broken down to reading an input, pausing a read, stopping a read, and resuming a read. Additionally, it would become even more difficult if these asynchronous events needed to be mixed with synchronous events. Node is built to handle all other tasks until an asynchronous event has completed execution. For example, if a file is being read-in while the information from the file is being manipulated, if not handled correctly—such as using an event catcher to manipulate the file only after reading has completed—the manipulation will occur before the file has been read completely. The problem that encompassed all these issues was problem 1—unique words. This problem focused on being given information from the user about a file, reading the file and then printing information about that file to the user based on their specifications. This problem consists of many different events that all must be handled in a specific order to work properly for the outcome. I had to solve this even by using several different events catching mechanisms as well as exploiting some synchronous features of Node using methods/ functions calls during asynchronous events.

Work Cited

Node.js Foundation, & Joyent. (n.d.). Node.js v8.11.1 Documentation. Retrieved May 2, 2018, from https://nodejs.org/dist/latest-v8.x/docs/api/

Syed, B. A. (2014). *Beginning Node.js*. Berkeley, CA: Apress.