Node.js Training

**The front end**

* A web server is a process running on a computer that listens for incoming requests for information over the internet and sends back responses.
* Dynamic content caters to a specific user rather than sending the same files to every visitor
* Databases can be divided into two types:
  + Relational: store information in tables with columns and rows
  + Non-relational (NoSQL): uses something like key-value pairs or a document storage model.
* SQL = Structured Query Language
* API = Application Program Interface
* API’s are generally used to create, read, update or delete database information
* Authentication is the process of validating the identity of a user, eg password and username. Credential will be stored on the back end of a database.
* Authorization controls which users have access to which resources and actions.
* Front end is developed through HTML, CSS and JavaScript.
* Backend can be developed through multiple language, like PHP, Java, JavaScript, Python and more.
* Most developers use frameworks to work with the back end.
* MEAN Stack
  + MongoDB
  + Express.js
  + Angular.js
  + Node.js
* LAMP stack
  + Linux
  + Apache
  + MySQL
  + PHP
* Node.js is a JavaScript runtime, or an environment that allows us to execute JavaScript code outside of the browser.
* REPL =
  + Read: program reads input from a user
  + Eval: program evaluates the user input
  + Print: program prints the evaluation to the console
  + Loop: loops through the steps again.
* To type multiple lines, hit **.editor** to input lines without evaluation. To exit editor, use **ctrl + D**.
* Global Objects
  + Global:
  + Process:
  + Buffer:
  + clearImmediate:
  + clearInterval:
  + clearTimeout:
  + setImmediate:
  + setInterval:
  + setTimeout:
  + console:
  + module:
  + require:
* global at its core is a JavaScript object
* a process is an instance of a computer program being executed.
* **Process.env** is an object which stores and controls information about the environment in which the process is currently running.
* The **process.memoryUsage()** returns information on the CPU demands of the current process. Adding .heapUsed will return a number stating how many bytes of memory the current process is using.
* The **process.argv** will show an array of command line values provided when the process was initiated.
* Modularity is a software technique where function is assigned to distinct parts.
* Modules are accessed through the **require()** function
* Core modules are modules included in the environment.
* Require() will first check if a module is a core module, and then will move on if not.
* Npm = node package manager, which is an online collection of software.
* Nodemon automatically restarts node applications when file changes in the directory are detected.
* Event-driven architecture = logic that can handle an event when it occurs, without knowing a time.
* Node uses **EventEmitter**,
  + which has an **.on()** method that assigns a listener callback function. The argument is the name of the event as a string, and the listener callback function as the second argument.
  + It also has an .**emit()** method, that announces a named event has occurred. The first argument is the name of the event as a string, and the second argument, the data is passed back into the listener callback function.
* Node.js uses event loops to queue actions. After an API performs a task, the callback information is turned to a queue to be processed.
* Input is data provided to the program, output is what the program returns.
* In node, **stdin** is used for input, **stdout** is used for output.
* The **buffer** objects are used to represent a fixed-length sequence of bytes.
* Many asynchronous node APIs just error first callback functions, where the error is the first expect argument and the data is the second.
* All the data on a computer is organized and access through a filesystem. Scripts should only have limited access to a user’s filesystem. This helps protect against malicious programs and invasions of privacy.
* Isolating applications and data is known as sandboxing.
* The **fs** core module is an API for interacting with the file system.
  + Methods available through the fs module have a synchronous version and an asynchronous version.
  + The **.readFile()** method reads data from a provided file.
* We don’t usually want entire files, but we want them piece by piece
* We can use **.createInterface()**  from the **readline** core module.
  + Examples of calling/requiring core modules.
  + Const readline = require(‘readline’)
  + Const fs = require(‘fs’)
* We can read from a stream, but we can also write to a stream.
* Node can create an HTTP server using **http.createServer()**.
* The http server has a method called .**listen()**, which watches for incoming connections.