1 Build a Back-End with Node/Express.js

1.1 Introduction

1.2 Node REPL

- Is an abbrebivation for Read-eval-print loop
- Node comes with built-in javascript REPL
- `.editor` goes into editor mode
  - Use `CTRL + D` when ready to evaluate the input
- A REPL can be extremely useful for performing calculations
- The Node environment contains a number of Node-specific `global` elements in addition to those built into the JavaScript language
  - can be examined using command `console.log(global)`

1.3 Running a Program with Node

- Done using command `node myProgram.js`
- Javascript code is written to file `.js` extension

1.4 Accessing the Process Object

- Node has a global `process` object with useful methods and information about the current process.
  - `process.env` property is an object which stores and controls information about the environment in which the process is currently running
    - `PWD` - holds a string with the directory where the current process is located
    - `NODE_ENV` - holds a value of either production or development

    Example

```
1    if (process.env.NODE_ENV === 'development'){
2        console.log('Testing! Testing! Does everything work?');
3    }
4
```

    - `process.memoryUsage()` returns information on the CPU demands of the current process.
    - `process.memoryUsage().heapUsed` return a number representing how many bytes of memory the current process is using.

- `process.argv` property holds an array of command line values provided when the current process was initiated
  - first element in the array is the absolute path to Node
  - second element in the array is the path to the file that's running
  - following elements will be any command line arguments provided when the process was initiated (like C)!!!

```
node myProgram.js testing several features

```

```
console.log(process.argv[3]); // Prints 'several'

```

1.5 Core Modules and Local Modules

- **Modularity** is a software design technique where one program has distinct parts each providing a single piece of the overall functionality.
  - Is essential when creating scalable programs
    - incorporate libraries and frameworks and separate the program's concerns into manageable chunks
- **Modules** come together to build a cohesive whole
  - is imported using `require()`

```
// Require in the 'events' core module:
let events = require('events');

```

  - is exported using `module.exports`

```
module.exports = class Dog {

    constructor(name) {
        this.name = name;
    }

    praise() {
        return `Good dog, ${this.name}!`;
    }
};

```

1.6 Node Package Manager

- NPM, which stands for Node Package Manager

1.7 Event-Driven Architecture

- Node is often described as having event-driven architecture
- Node provides an `EventEmitter` class which we can access by requiring in the events core module

```
1    // Require in the 'events' core module
2    let events = require('events');
3
4    // Create an instance of the EventEmitter class
5    let myEmitter = new events.EventEmitter();
6
```

1.8 Event-Driven Architecture

- Node is often described as having event-driven architecture.
  - This feels so much like threaded programming
- event emitter instance has an `.on()` method which assigns a listener callback function to a named event.
  - first argument - the name of the event as a string
  - second argument - the listener callback function

```
1
2    let newUserListener = (data) => {
3        console.log(`We have a new user: ${data}.`);
4    };
5
6    // Assign the newUserListener function as the listener callback for '
     new user' events
7    myEmitter.on('new user', newUserListener)
8
9    // Emit a 'new user' event
10   myEmitter.emit('new user', 'Lily Pad') //newUserListener will be
     invoked with 'Lily Pad'
11
```

1.9 Asynchronous JavaScript with Node.js