

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

## 1.1 Introduction

## 1.2 Node REPL

- Is an abbreviation 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)!!!

```
1 node myProgram.js testing several features
2
```

```
1 console.log(process.argv[3]); // Prints 'several'
2
```

## 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()`

```
1 // Require in the 'events' core module:
2 let events = require('events');
3
```

- is exported using `module.exports`

```
1 module.exports = class Dog {
2
3     constructor(name) {
4         this.name = name;
5     }
6
7     praise() {
8         return `Good dog, ${this.name}!`;
9     }
10 };
11
```