# Node.js

Invented in 2009

Original author: Ryan Dahl

Now maintained by Node.js Foundation

Node.js is a trademark of Joyent, Inc.

# What is Node.js?

- Node.js is an open-source, **cross-platform** runtime environment for developing **server applications**

- Node.js applications are written in **JavaScript**
  - You may choose to write in TypeScript

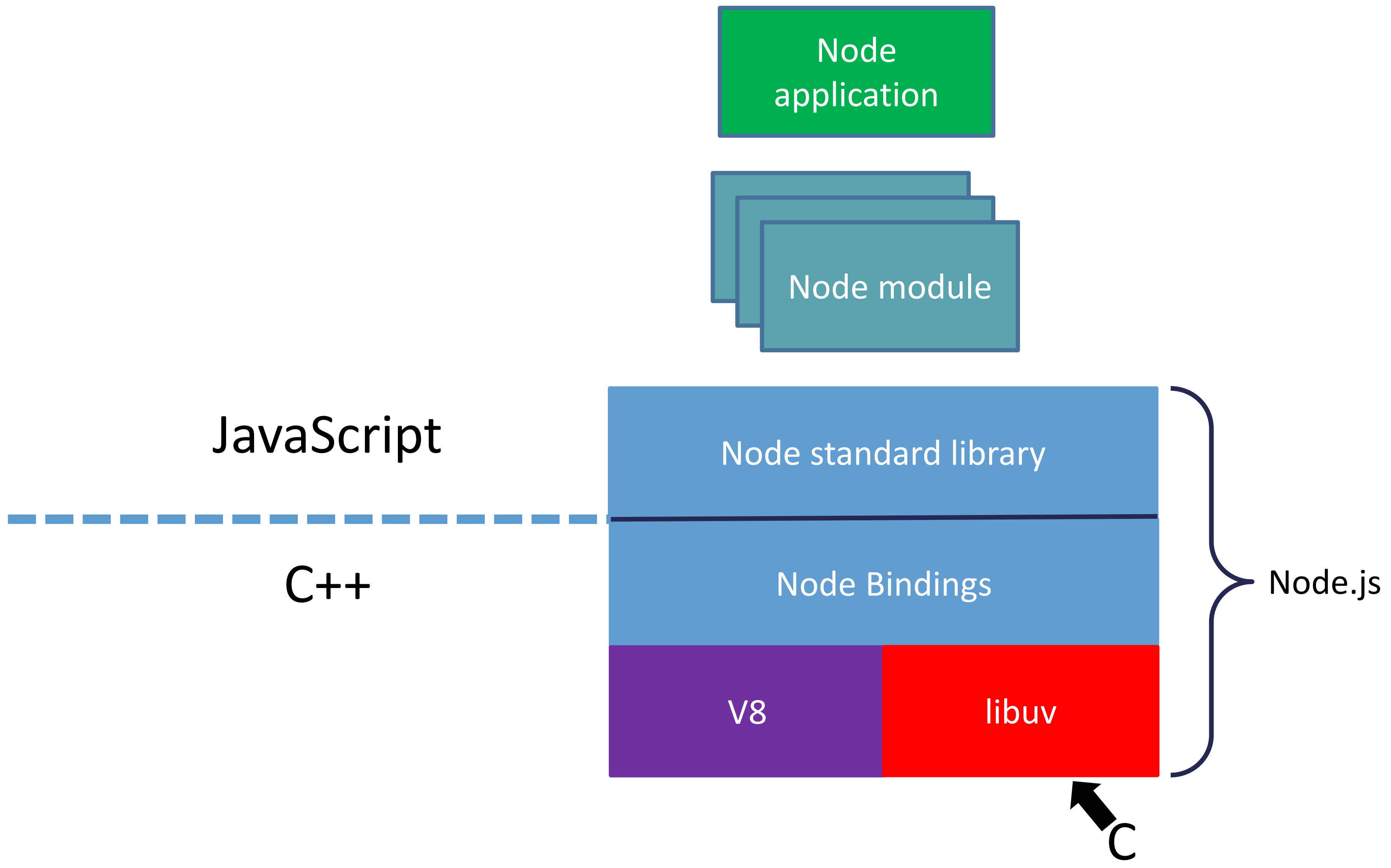
- Node.js provides an **event-driven architecture** and a non-blocking I/O API designed to optimize an application's throughput and scalability for real-time web applications

- Node.js contains a built-in library to **allow applications to act as a web server** without software such as Apache HTTP Server, Nginx or IIS
  - But Nginx is often used as a proxy server in front of Node

## Why Use Node?

- When coded correctly, it’s fast and makes very efficient use of system resources
- Full stack development
  - Use of JavaScript on both the server and client makes life easier for full stack developers

## Node Internals



## libuv

- A library written in C
- Enforces an **asynchronous**, **event-driven** style of programming
- Its core job is to provide an event loop and callback based notifications of I/O and other activities
- Offers core utilities like timers, non-blocking networking support, asynchronous file system access, child processes and more
- Pseudocode:

```
while there are still events to process:
  e = get the next event
  if there is a callback associated with e:
    call the callback
```

- Examples of events:
  - File is ready for writing
  - A socket has data ready to be read
  - A timer has timed out

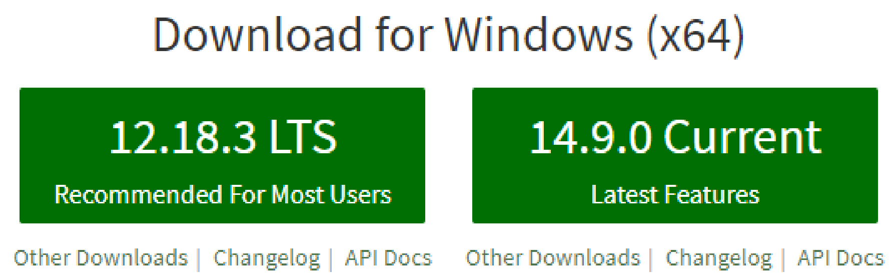
## Node Fundamentals

- With the single-threaded model it’s important to remember that all of your clients use the same central process
- To keep the flow smooth you need to make sure that nothing in your code causes a delay, blocking another operation

  → **You must use asynchronous programming to access the database etc**.

## Installing Node.js and npm

- Windows and Mac:
  download an installer from the Node.js website:
  https://nodejs.org/en/


Download for Windows (x64)

12.18.3 LTS
Recommended For Most Users

14.9.0 Current
Latest Features

Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

- Linux (Ubuntu):
  - Debian and Ubuntu based Linux distributions, Enterprise Linux/Fedora and Snap packages
  - Node.js binary distributions are available from NodeSource.
  - For complete instructions on how to install:
    https://nodejs.org/en/download/package-manager/

## Verifying installation

- Once you have Node and npm installed you can check the versions you have with a couple of terminal commands:

```
node --version
npm --version
```

*Shortcuts*

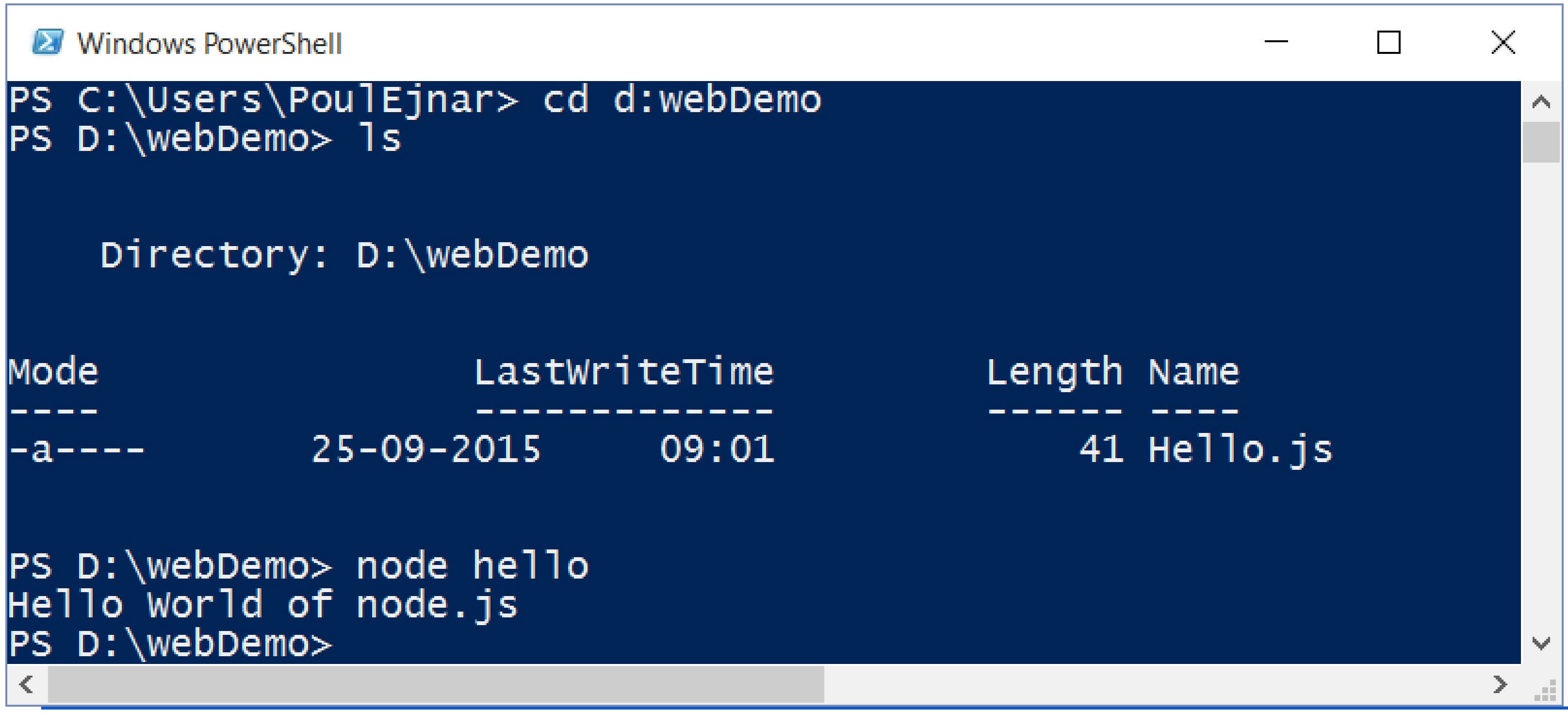
```
node -v
npm -v
```

## Hello World

- Open an code editor of your choice and enter the JavaScript code:

  ```
  console.log('Hello world of node.js');
  ```

- Save the file as Hello.js
- Open a command prompt / powershell / terminal
  Move to the folder with the source file and let node execute the file:



```
PS C:\Users\PoulEjnar> cd d:webDemo
PS D:\webDemo> ls


    Directory: D:\webDemo


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       25-09-2015     09:01             41 Hello.js


PS D:\webDemo> node hello
Hello World of node.js
PS D:\webDemo>
```

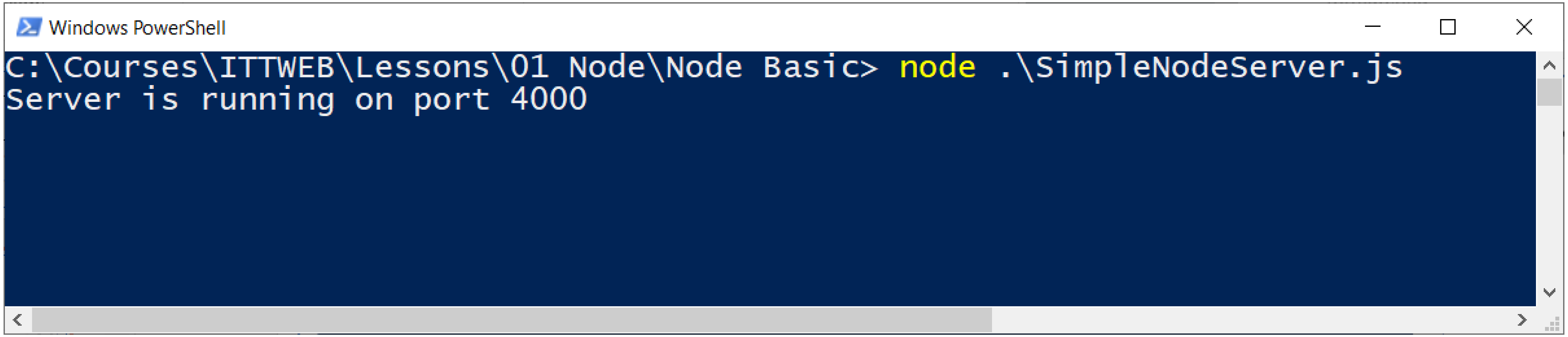
AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

# A Simple Node Server

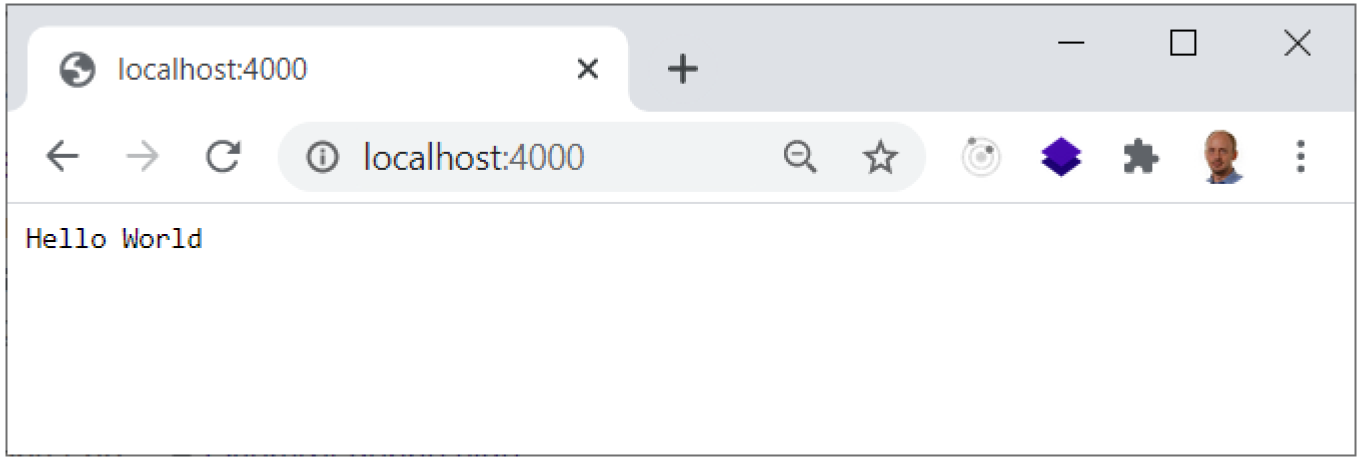
```
var http = require('http');
var port = process.env.PORT || 4000;
var server = http.createServer(function(req, res) {
   res.writeHead(200, { 'Content-Type': 'text/plain' });
   res.end('Hello World\n'); });
server.listen(port)
console.log('Server is running on port ' + port);
```

- Use the http object to create an HTTP server
  - The parameter is a function literal that takes a req (HTTP request) object and a res (HTTP response) object as parameters
  - and uses res to write the HTTP response back
- This idiom is ubiquitous throughout all levels of the Node.js stack

## Running The Simple Node Server



```
C:\Courses\ITTWEB\Lessons\01 Node\Node Basic> node .\SimpleNodeServer.js
Server is running on port 4000
```



```
Hello World
```

**Remember to end the server task!**
**With ctrl-c**

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

## Common error

- If you get this error:
  - **Error: listen EACCES: permission denied** 0.0.0.0:3000

- Then use a port number > 3016
  - Or use port 80  or 443

## npm

- A package manager was introduced for Node.js in 2011
- npm is installed automatically with Node
- Allows publishing and sharing of open-source Node.js libraries by the community
- Is designed to simplify installation, updating and uninstallation of libraries aka. **Node modules**

- Thousands of open-source libraries have been built for Node.js, most of which are hosted on the npm website

## Use of npm

- npm modules are retrieved over the Internet from the public package registry maintained on http://npmjs.org
- Modules may be installed through npm install

```
npm install moduleName
```

- To install a module/tool globally use –g
  - In a shell as administrator (sudo)

```
npm install moduleName -g
```

## package.json

- In every Node application there should be a file in the root folder of the application called `package.json`
  - contains metadata about the project
  - references the packages that it depends on
- Use the **`npm init`** command to create a package.json file for your application
  - This command will prompt your for a number of things such as the name and version of your application

```
{
  "name": "mean",
  "version": "0.1.0",
  "description": "Demo project",
  "main": "index.js",
  "scripts": {
    "test": "test”
  },
  "author": "Poul Ejnar",
  "license": "ISC"
}
```

AARHUS UNIVERSITY
SCHOOL OF ENGINEERING

## To Add a Module

- To add the Express module to a Node project:

```
npm install express --save
```

*Node modules installed with the --save option are added to the dependencies list in the package.json file*

```
{
  "name": "mean",
  "version": "0.1.0",
  "description": "Demo project",
  "main": "index.js",
  "scripts": {
    "test": "test”
  },
  "author": "Poul Ejnar",
  "license": "ISC",
  "dependencies": {
    "express": "^4.13.3"
}
```

You may use the short form:
**`npm i express -S`**

*Use newest patch*

## Handling Page Requests

- Examine the request object's url attribute
  - If it matches with your expectations, then return the associated page

```
var http = require('http');
var port = process.env.PORT || 4000;
var server = http.createServer(function (req, res) {
    if (req.url == '/') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.write('Welcome to http nodejs');
        res.end();
    } else
    if (req.url == '/customer') {
        res.writeHead(200, { 'Content-Type': 'text/plain' });
        res.write('Welcome to Customer page');
        res.end();
    } else {
        res.writeHead(404, { 'Content-Type': 'text/plain' });
        res.write('Page not found');
        res.end();
    }
});
server.listen(port)
console.log('Server is running on port ' + port);
```

## Socket Programming

- You can create an tcp socket application by use of the **`net`** module
  - Or a UDP socket with the **`dgram`** module

```
var net = require('net');
var server = net.createServer(function(client) {
    console.log('client connected');

    // Waiting for data from the client.
    client.on('data', function(data) {
        console.log('received data: ' + data.toString());
        // Write data to the client socket.
        client.write('hello from server');
    });
    // Closed socket event from the client.
    client.on('end', function() {
    console.log('client disconnected');
    });
});
```

## How to Create a Module

- Use the exports object to export funktions and classes from a JavaScript file

```
// MyModule.js
var calculate = function(numA,numB){
    return numA*numB + 10*numB;
}

var add = function(numA,numB){
    return numA + numB;
}

var sub = function(){
    return numA - numB;
}

exports.calculate = calculate;
exports.add = add;
exports.sub = sub;
```

```
// UseModule.js
var myModule = require('./MyModule.js');
var result = myModule.calculate(20, 10);
console.log(result);
result = myModule.add(2, 3);
console.log(result);
```

## How to Export a Class?

- Just export the Constructor

```
// constructor
var MyClass = module.exports.MyClass = function () {
    console.log("In MyClass' constructor");
    this.size = 42;
}
// methods
MyClass.prototype.bar = function (a, b) {
    console.log('In bar - ' + a + '-' + b);
}
// Properties
Object.defineProperty(MyClass.prototype, "size", {
    get: function () {
        console.log('In getter');
        return size;
    },
    set: function (value) {
    console.log('In setter');
    size = value;
    }
});
```

```
// UseModule.js
var myModule = require('./MyModule.js');
var obj = new myModule.MyClass();
obj.bar(8, 9);
console.log(obj.size);
obj.size = 47;
console.log(obj.size);
```

## Use of ES Modules

- The default module system in Node is CommonJS (require).
- But support for ECMAScript modules is enabled by default.
- Node.js will treat the following as ES modules when passed to node as the initial input, or when referenced by import statements within ES module code:
  - Files ending in .mjs.
  - Files ending in .js when the nearest parent package.json file contains a top-level field "type" with a value of "module".

```
// package.json
{
  "type": "module"
}
```

## File endings for modules

- Files ending in .js are still treated as CommonJS modules.
- Files ending in ***.mjs*** are explicitly treated as ES modules in *import* statements.

- You can turn this around if you add `"type": "module"` to the *package.json* for your project,
  - **Then Node will treat all *.js* files in your project as ES modules**.
  - If some of your project's files use CommonJS and you can't convert your entire project all at once, you can either rename those files to use the **.cjs extension**
  - Or put them in a subfolder containing a *package.json* with { "type": "commonjs" }, under which all *.js* files are treated as CommonJS modules.

## Node with Typescript

- TypeScript is a typed (optional) super-set of JavaScript that can make it easier to build and manage large-scale JavaScript projects.
- It can be thought of as JavaScript with additional features such as strong **static typing**, compilation and object oriented programming etc.

  → Better intellisence in VS Code!

- Setting Up a Node Project With Typescript
  – Follow this tutorial: https://scotch.io/tutorials/setting-up-a-node-project-with-typescript

## References & Links

- Get Programming with Node.js, by Jonathan Wexler
- https://en.wikipedia.org/wiki/Node.js

- **Node.js Best Practices** ☆☆☆
  https://github.com/i0natan/nodebestpractices

- libuv
  http://nikhilm.github.io/uvbook/basics.html
- **Node.JS Module Patterns -** A simple introduction to *Node*.JS *modules*
  *https://darrenderidder.github.io/talks/ModulePatterns/*