

Industry 4.0 Academy

Introduction to MEAN Stack

Introduction to Module 2

- Advanced Javascript
- Backend Development
- Database (Relational vs Non Relational vs Graph DB)
- MEA(R)N Stack : MongoDB, Express JS, Angular/React, Node.JS
- Cloud
 - AWS
 - Google Cloud Platform
 - Azure
- Devops
 - Continuous Deployment
 - Continuous Integration
 - Docker
- System to system connection
 - Predictive Learning?
 - Recommendation based on Weather?

What are we going to build?

- REST API
- Intelligent REST API :) -> **Authentication, Security**
- System connecting to other system
- Connecting the system to your frontend (Using VueJS, and later using React.js)

Module 2 class structure.

- Class... Discover new things! :)
- Exercise
- Code Challenge. - Application.. (50-50)
- Project...

Module 2 Project Requirement.


Booking API (virtual coach API, school material management, quiz API)...

- Dynamic data from backend.
- CMS for admin to manage content.
- Login/Authentication function.
- Intelligent features within the API, eg: Recommendation, Sending Email, Sending push notification..
- Aggregate data from different website.

Application and Data

Application and Data

Sponsored



Datadog
See metrics from all of your apps, tools and services in...

[Visit Website](#)

TOP 10 TOOLS & SERVICES

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
									
	JavaScript								

STACK LAYER

Application and Data

Utilities


DevOps

Business Tools

DevOps

DevOps

Sponsored



Datadog
See metrics from all of your apps, tools and services in...

[Visit Website](#)

TOP 10 TOOLS & SERVICES

1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
									

STACK LAYER

Application and Data

Utilities

DevOps

Business Tools

4 categories in a stack (Airbnb)

- Database - MySQL, Redis, RDS
- Frontend Framework - ReactJS
- Operating System/ Runtime environment - EC2 (Linux)
- Web Server - Nginx

4 categories in a Stack (Uber)

- Database - Postgresql, MongoDB, Redis, AresDB
- Frontend Framework - jQuery, React
- Operating System/ Runtime environment - EC2
- Web Server - Nginx

LAMP Stack

Database - MySQL

Frontend Framework - PHP, Perl, Python

Operating System/ Runtime environment - Linux

Web Server - Apache



Content Management System

- Wordpress
- Drupal
- Magento
- WooCommerce by Wordpress
- Joomla
- Zencart
- Shopify

MEAN STACK



Mongo DB
(database system)



Express
(back-end web
framework)

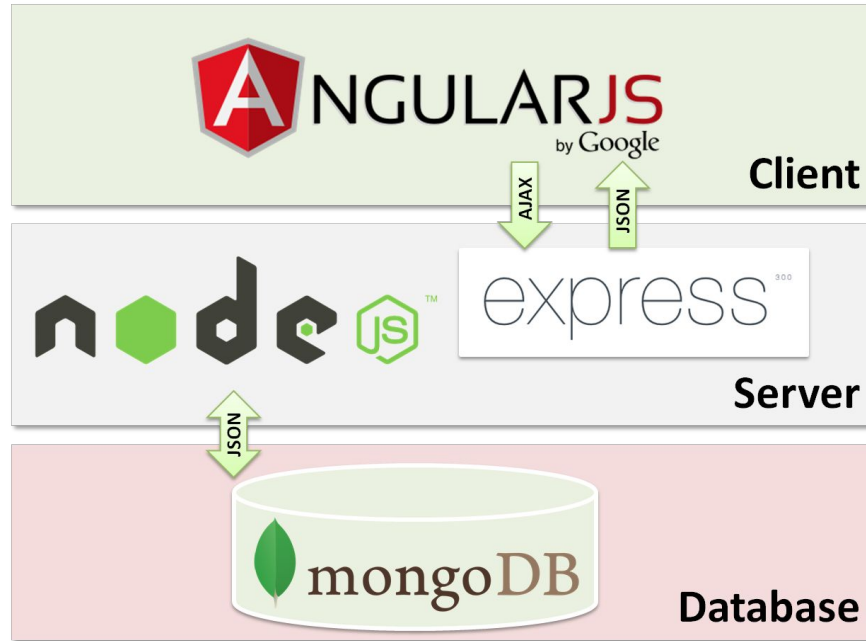


Angular.js
(front-end
framework)



Node.js
(back-end runtime
environment)

What is ME(A/R/V)N?



MEAN is an opinionated fullstack javascript framework - which simplifies and accelerates web application development.

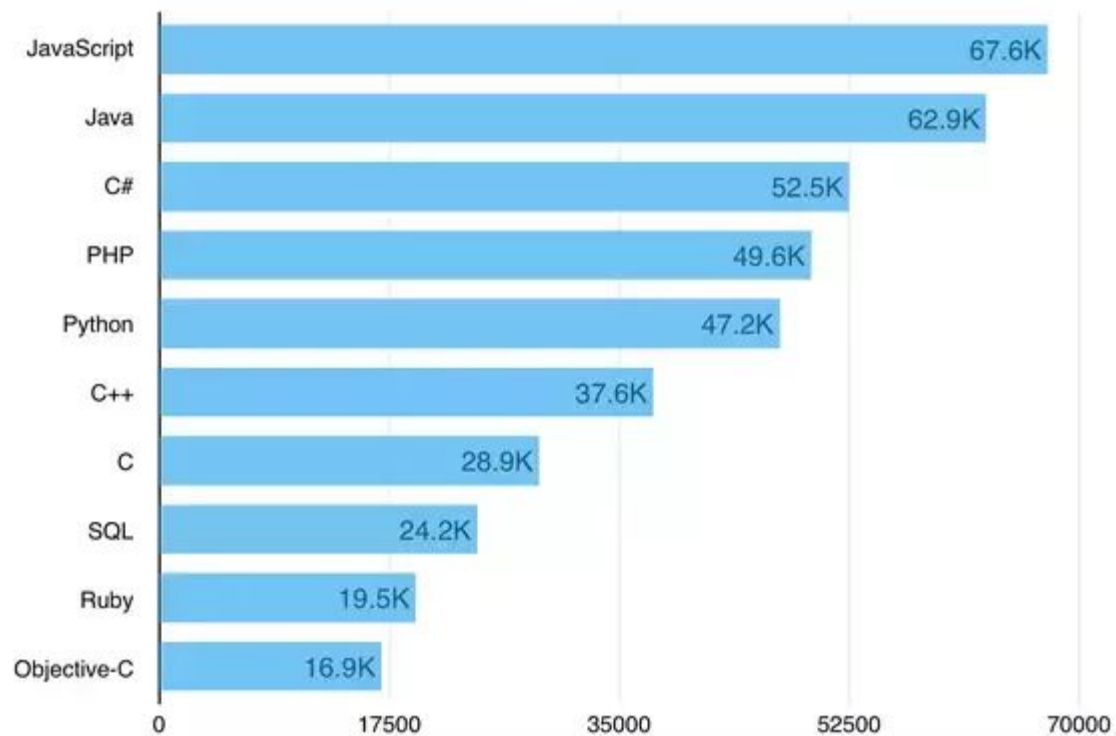
Why MEAN?

- 100% Free
- 100% Open Source
- 100% (Javascript and HTML)
- Single language throughout the application. (Javascript)
- Adhere to MVC concept.
- Use of JSON as data structure, compared to before where serialization and deserialization of data structure is needed.

The fullstack journey

	JS	Python	PHP (server side language/ framework)	Java - Normally used in enterprise application	.Net (C#)
Frontend web framework	React.JS/VueJS/Angular	Django/Flask	Laravel/CodeIgniter/Yii	Angular	.Net
Database	MongoDB	Mysql, Postresql	MySql	Oracle	MS SQL
Runtime environment	NodeJS	Linux	Linux	Tomcat/Jboss (Java Server)	.Net
Web server	ExpressJs	Django/Flask	Laravel/CodeIgniter/Yii	Spring/SpringBoot	.Net
Mobile app (module 3)	React Native/ Ionic	Cannot	Cannot	Android Native	Xamarin
Desktop app	Electron	Tkinter	Cannot	Yes JSF	??
Data science	Can do it but no mainstream	Main language for data science	Cannot	cannot..	??

StackOverflow Tag Followers (as of April 2015)



Node JS

- Open Source, cross platform runtime environment for server side and networking application.
- Written in Javascript and can run on Linux, Mac, Windows and FreeBSD running on the server.
- Based on event-driven architecture, and non blocking I/O that and optimize and scalability.
- Uses Google Javascript V8 Engine to execute code.
- Used by Groupon, SAP, LinkedIn, Microsoft, Yahoo, Walmart, Paypal.

Why uses NodeJS?

- It is very lightweight and fast.
- Easy to configure.
- A lot of module available for free.
- Works with SQL and NoSQL.

What can be build with Node.js?

- REST API & Backend Applications
- Real-Time Services (Chat and Games)
- Blogs, CMS and Social Application.
- Utilities and Tools.
- Anything that is non CPU intensive

[HOME](#)[ABOUT](#)[DOWNLOADS](#)[DOCS](#)[FOUNDATION](#)[GET INVOLVED](#)[SECURITY](#)[NEWS](#)

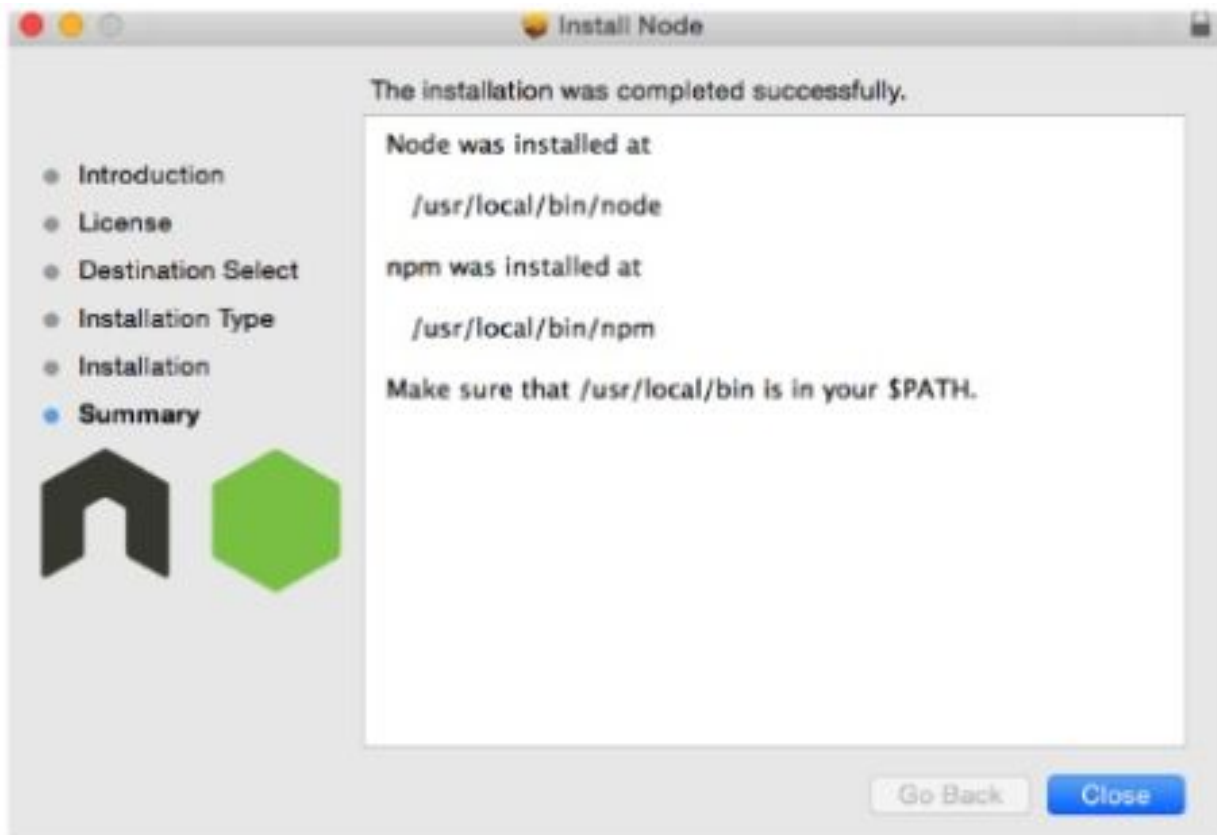
Join us for Node.js Interactive
happening in Vancouver, Canada
October 4 - 6, 2017

Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#). Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, [npm](#), is the largest ecosystem of open source libraries in the world.

<http://nodejs.org>

Installing NodeJS

- 1) Download Node.JS from nodejs.com
- 2) Choose the required nodejs version.
- 3) Run the downloaded .msi file.
- 4) Open powershell and enter `npm -v` and `node -v` to verify it has been installed.



Creating First Application: Hello Node!

Create a new file index.js and add the following line of code.

```
var http = require('http');

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello Node!');
}).listen(8080);

console.log('Server started on localhost:8080; press Ctrl-C to terminate....');
```

Open terminal/CMD and type the following command to run the server :
node index.js

Internet - Interconnection of network..

Server - Computer, that is used to host data or information

Web Server - Computer, that is used to host a website

IP Address - An address that each of the device have to be connected to each other..

Port - > is a number that links an application so that i can be connected from outside .. 192.168.0.115 -> 32932

Node HTTP Module

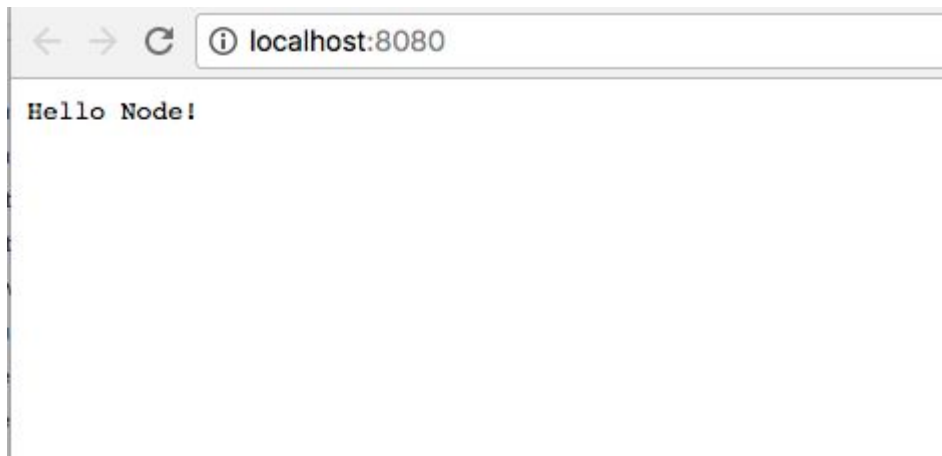
The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).

How to run a Node program?

- 1) Open terminal or CMD.
- 2) Open the folder where the project is.
- 3) Type node 'filename' to complete.

```
index.js
MBP:helloworld Me-Tech$ node index.js
Server started on localhost:8080; press Ctrl-C to terminate....
█
```



npm

npm stands for Node Package Manager or NodeJS package manager.

It is the default package manager for Node.js.

It runs through the command line and it makes it easier to specify and link dependencies.



find packages



sign up or log in



Build amazing things

npm is the package manager for JavaScript and the world's largest software registry. Discover packages of reusable code — and assemble them in powerful new ways.

Sign up for npm



npm important command

Command	Description
<code>npm --version</code>	Verify the version of npm installed inside
<code>npm install <module name></code>	Installing a new module inside project or globally
<code>npm uninstall <module name></code>	Uninstall a module
<code>npm update <module name></code>	Updating a module.
<code>npm search <module name></code>	Search for a module in npm repo.
<code>npm init</code>	Start creating a module.
<code>npm publish</code>	Publishing a module.



Using nodemon module

Using Nodemon

Monitor for any changes in your node.js application and automatically restart the server - perfect for development

```
npm install -g nodemon
```

When you run the application, instead of using `node <application name>` you will use

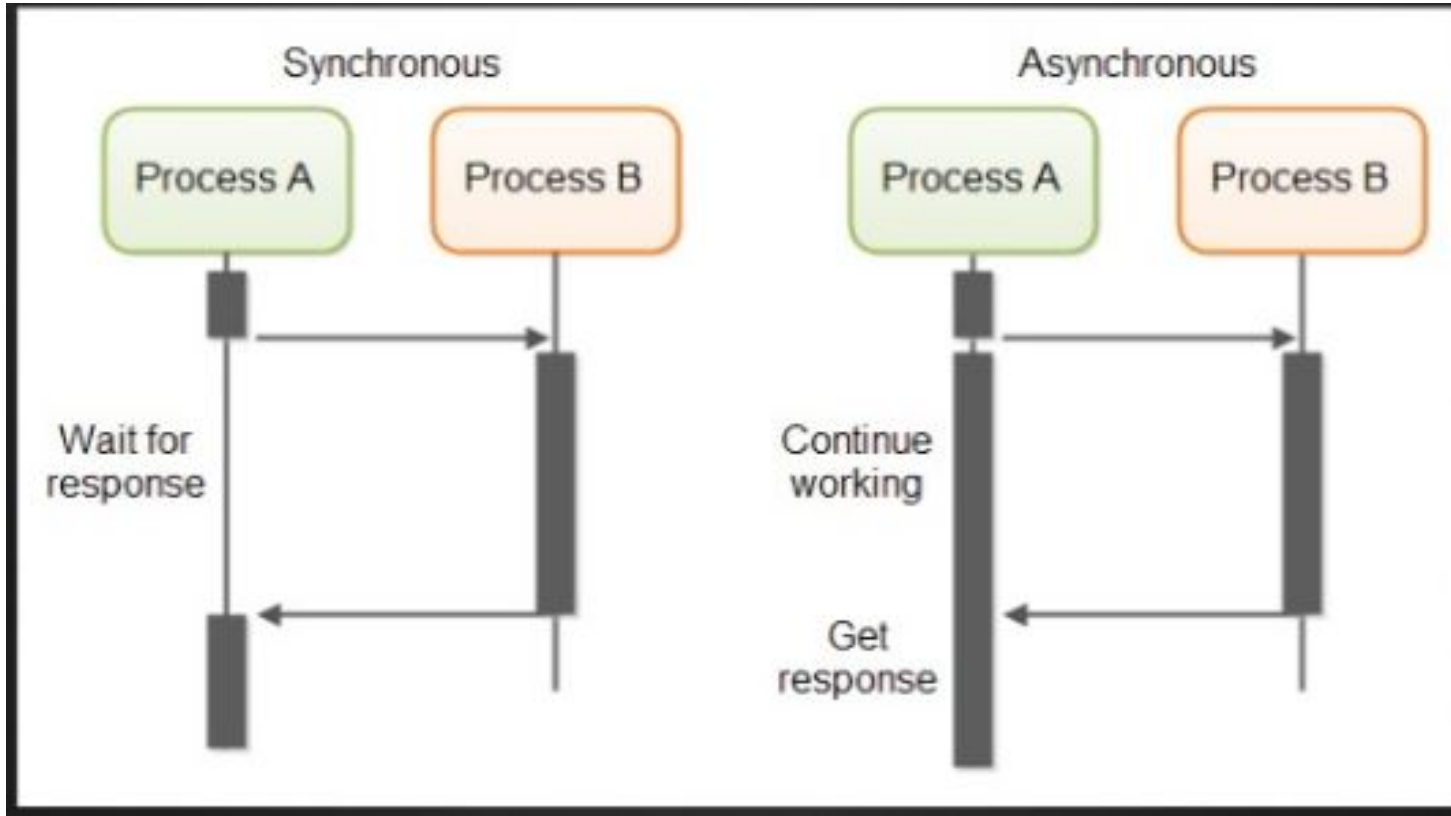
```
nodemon [your node app]
```

Whenever a change of code is detected, your local server will be restarted automatically.

Try It: Downloading and Using a package

- 1) Go to <http://www.npmjs.com> and look for package to transform string from lowercase .
- 2) Use npm command to install the package.
- 3) Add the package into the your 'Hello World' project.
- 4) Change the casing of the letter to be uppercase everywhere.

Synchronous vs Asynchronous programming



Synchronous vs Asynchronous programming

In *synchronous* programs, if you have two lines of code (L1 followed by L2), then L2 cannot begin running until L1 has finished executing.

In *asynchronous* programs, you can have two lines of code (L1 followed by L2), where L1 schedules some task to be run in the future, but L2 runs before that task completes.

Try It: What will be the result?

```
function getData() {  
  var data; $.get("https://www.randomuser.me/api",  
  function(response) { data = response; });  
  return data; }  
  
var data = getData();  
  
console.log("The data is: " + data);
```

Callback in Node

- Callback is an asynchronous equivalent for a function.
- A callback function is called at the completion of a given task.
- Node makes heavy use of callbacks.
- All the APIs of Node are written in such a way that they support callbacks.
- Helps to maintain performance for single threaded NodeJS.

Example of non-callback function

```
var fs = require("fs");  
  
var data = fs.readFileSync('input.txt');  
  
console.log(data.toString());  
console.log("Program Ended");
```

Example of callback function

```
var fs = require("fs");

fs.readFile('input.txt', function (err, data) {
  if (err) return console.error(err);
  console.log(data.toString());
});

console.log("Program Ended");
```

Difference of non-callback and callback

When program running on non-callback block, the program is running in a sequence. In a programming point of view, it is easier to code however when a function takes a long time to be executed, it will block the main thread.

Callback function allows the program to continue even though the program is still fetching the data. Whenever the function finished execution its task, the callback function will be called.

Node FS Module

You may use Node.JS Filesystem (FS) Module if you need to do an operation for reading and writing from a filesystem.

This is a built in Node module that can be imported using **require('fs')**.

Every method in fs module has an synchronous and asynchronous methods. In case of asynchronous method, the last parameter is the completion function callback and the first method is the error.

Refer to <https://nodejs.org/api/fs.html> for documentation on Filesystem.

Try It: Global and File Module - Read From .txt

- 1) Refer to readFile documentation in Filesystem Node.JS documentation.
- 2) Create a Node.JS program that will open the previously created file and append at the end of the line 'Adding new text here..'
- 3) Create the program using:
 - a) Asynchronous execution
 - b) Synchronous execution

Try It: File Module - Read from file.

- 1) Create a .txt file inside a new folder within your project. Create "Hello World" inside the .txt file.
- 2) Retrieve from index.js and retrieve the text inside the .txt file.

Try It: File Module - Write and read from file.

- 1) Refer to readfile documentation in Filesystem Node.JS documentation.
- 2) Create a Node.JS program that will open a .txt file and write "Hello World" to a new file.
- 3) Create the program using:
 - a) Asynchronous execution
 - b) Synchronous execution
- 4) Open the file that has been created and show the items in the file on console.log

Try It: File Module - Append from file.

- 1) Refer to readfile documentation in Filesystem Node.JS documentation.
- 2) Create a Node.JS program that will open the previously created file and append at the end of the line 'Adding new text here..'
- 3) Create the program using:
 - a) Asynchronous execution
 - b) Synchronous execution

Callback Hell / Pyramid of Doom

Callback Hell is the situation where the syntax of a callback is nested into several level.

Callback Hell causes code confusion and make the code difficult to understand.

It can also be referred as pyramid of doom.

```
doAsync1(function () {  
  doAsync2(function () {  
    doAsync3(function () {  
      doAsync4(function () {  
      })  
    })  
  })  
})  
})
```

Example of a Callback Hell code

```
var fs = require('fs');

var myFile = '/tmp/test';
fs.readFile(myFile, 'utf8', function(err, txt) {
  if (err) return console.log(err);

  txt = txt + '\nAppended something!';
  fs.writeFile(myFile, txt, function(err) {
    if(err) return console.log(err);
    console.log('Appended text!');
  });
});
```

Avoiding Callback Hell

- 1) Create different functions for different callback.

```
function appendText(txt, err){
  if(err) return console.log(err);
  txt = txt + '\nAppended something!';
}

function notifyUser(err){
  if(err) return console.log(err);
  console.log('Appended text!');
}
```


Avoiding Callback Hell

2) Call the function in each other as callback.

```
var fs = require('fs');

function notifyUser(err) {
  if(err) return console.log(err);
  console.log('Appended text!');
};

function appendText(err, txt) {
  if (err) return console.log(err);

  txt = txt + '\nAppended something!';
  fs.writeFile(myFile, txt, notifyUser);
}

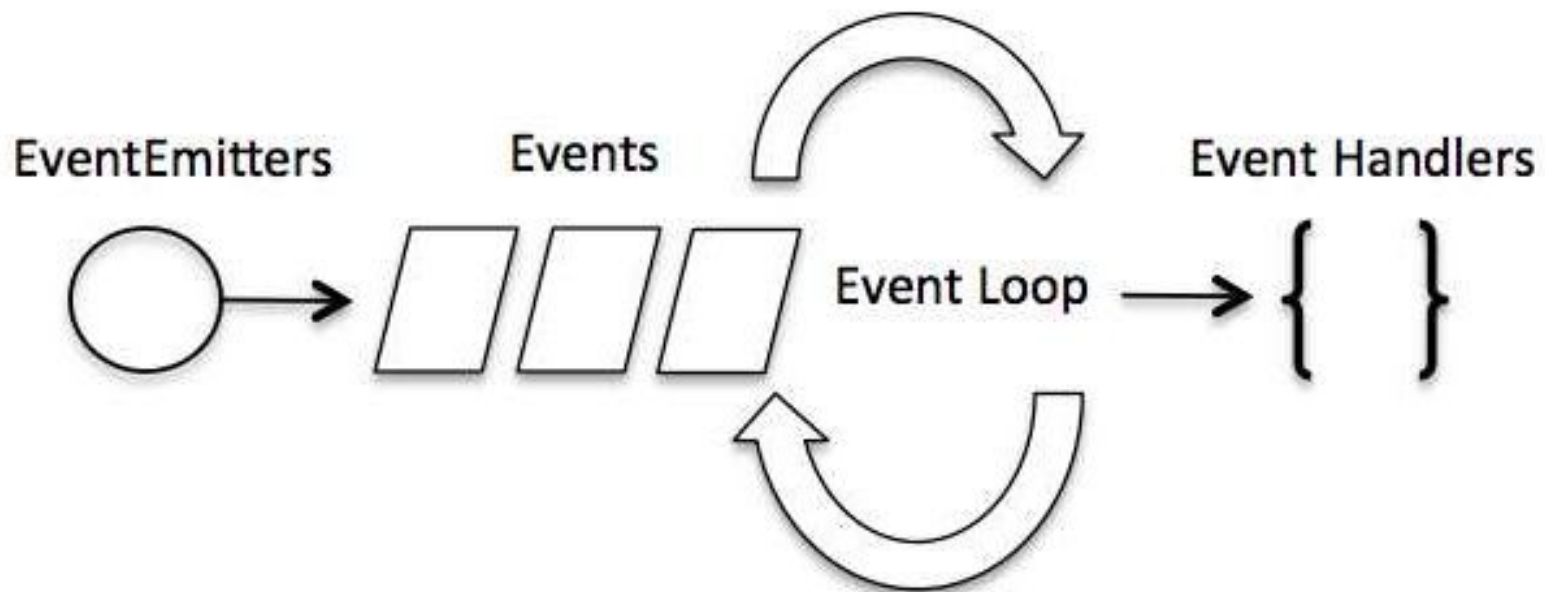
var myFile = '/tmp/test';
fs.readFile(myFile, 'utf8', appendText);
```

Event Driven Programming

A programming paradigm: When an event occurs, code something to respond to them.

Eg: After a mouse click, when the keyboard is touched (Seen in Module 1)

In previous example, the event `createServer` takes a function as argument that will be invoked every time HTTP request is invoked. The function will return a simple text of Hello World.



Events in Node

- Support Node.JS to maintain concurrency.
- Node thread keeps an event loop.
- It fires corresponding event when the task is completed.
- Event signals event listener
- The difference between events and callback is that callback is called when an asynchronous function is being completed, whereas event handling works with Observer pattern.

Steps to create Event in Node.

- 1) Use ***events*** module.
- 2) Create an event emitter : `var emitter = new events.EventEmitter();`
- 3) Create the event handler : `eventEmitter.on('eventName',
eventHandler);`
- 4) Emit the event.

Events in Node

Methods	Description
addListener(event, listener)	Adds a listener at the end of the listeners array for the specified event.
on(event, listener)	Adds a listener at the end of the listeners array for the specified event. No checks are made to see if the listener has already been added.
once(event, listener)	Adds a one time listener to the event. This listener is invoked only the next time the event is fired, after which it is removed.
removeListener(event, listener)	Removes a listener from the listener array for the specified event.
emit(event, [arg1], [arg2], [...])	Execute each of the listeners in order with the supplied arguments.

Try It: Event

```
var events = require('events');  
var emitter = new events.EventEmitter();  
  
emitter.on("myEvent", function(){  
    console.log("Event Fired...");  
});  
  
emitter.emit("myEvent");
```

Try It: Event with parameter.

```
var events = require("events");
var emitter = new events.EventEmitter();

var username = "nodejs"
var password = "awsome"

emitter.on("userAdded", function(username, password){

console.log ("New user" + username);
});

emitter.emit("userAdded", username, password);
```


Error handling in Node

Whenever we write a callback function, you may notice that we have included **throw err** line of code.

This is call **'Error handling'** in Javascript. Whenever an error occur, we need to manage it to ensure correct message will be shown.

In the example below, in case the file is not there, the exception will be thrown:

```
fs.readFile('input.txt', function (err, data) {  
  if (err) throw err;  
});
```

Node Global Module

Function	Description
<code>__filename</code>	The filename of the current module. This is the resolved absolute path of the current module file.
<code>__dirname</code>	Get the directory name of the current module.
<code>setTimeout(callback, delay)</code>	global function is used to run callback after delay specified.
<code>clearTimeout</code>	Function to stop the timer that previously created.

Try It: Node Global Module.

- 1) Create a Node.JS script that will show in console the current directory of the file.
- 2) Get the current filename of the file.

Node HTTP Module

The Hypertext Transfer Protocol (HTTP) is an application protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web.

Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the HyperText Transfer Protocol (HTTP).

Routing

Routing is the mechanism for serving the client the content it has asked for.

In a web-based client/server applications, the client specifies the desired content in the URL; specifically, the path and querystring (the parts of a URL will be discussed in more detail

The server will then return the desired content to the client.

Try It: Routing

```
var http = require('http');
http.createServer(function(req,res){
// normalize url by removing querystring, optional
// trailing slash, and making it lowercase
var path = req.url.replace(/\/?(?:\?.*)?$/, "").toLowerCase();
switch(path) {
  case "":
    res.writeHead(200, { 'Content-Type': 'text/plain' }); res.end('Homepage');
    break;
  case '/about':
    res.writeHead(200, { 'Content-Type': 'text/plain' }); res.end('About');
    break;
  default:
    res.writeHead(404, { 'Content-Type': 'text/plain' }); res.end('Not Found');
    break;
}
}).listen(3000);
console.log('Server started on localhost:3000; press Ctrl-C to terminate....');
```

Server using Filesystem

```
var http = require('http'), fs = require('fs');

function serveStaticFile(res, path, contentType, responseCode) {
  if(!responseCode) responseCode = 200;
  fs.readFile(__dirname + path, function(err,data) {
    if(err) {
      res.writeHead(500, { 'Content-Type': 'text/plain' });
      res.end('500 - Internal Error');
    }
    else{
      res.writeHead(responseCode,
        { 'Content-Type': contentType });
      res.end(data);
    }
  });
}
```

Server using filesystem (2)

```
http.createServer(function(req,res)
{ // normalize url by removing querystring, optional // trailing slash, and making lowercase var
path = req.url.replace(/\/?(?:\?.*)?$/, "") .toLowerCase();
switch(path) {
case "": serveStaticFile(res, '/public/home.html', 'text/html');
break;
case '/about': serveStaticFile(res, '/public/about.html', 'text/html');
break; case '/img/logo.jpg': serveStaticFile(res, '/public/img/logo.jpg','image/jpeg');
break;
default: serveStaticFile(res, '/public/404.html', 'text/html', 404); break; } }).listen(3000);
console.log('Server started on localhost:3000; press Ctrl-C to terminate....');
```