

# Node.js

Harshita Maheshwari

# Agenda for Today's Session

- Introduction to Node.js
- JavaScript vs. Node.js
- why Node.js
- Event Loop
- Understanding Blocking I/O and Non-blocking I/O
- What can we do with Node?
- What can't do with Node?
- When to use Node
- When not use Node
- Node.js Environment Setup
- Node.js REPL



# Introduction

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications.

- **Written in C++**
- **created by Ryan Dahl starting in 2009**
- **Built on top of Chrome's V8 engine –so pure JavaScript**
- **Framework to build asynchronous I/O applications**
- **Single Threaded – no concurrency bugs –no deadlock**
- **One node process = one CPU core**
- **It is open source and free to use.**

# JavaScript vs. Node.js

JAVASCRIPT	NODEJS
Javascript is a programming language that is used for writing scripts on the website.	NodeJS is a Javascript runtime environment.
Javascript can only be run in the browsers.	NodeJS code can be run outside the browser.
It is basically used on the client-side.	It is mostly used on the server-side.
Javascript is capable enough to add HTML and play with the DOM.	Nodejs does not have capability to add HTML tags.
Javascript can run in any browser engine as like JS core in safari and Spidermonkey in Firefox.	Nodejs can only run in V8 engine of google chrome.
Javascript is used in frontend development.	Nodejs is used in server-side development.
Some of the javascript frameworks are RamdaJS, TypedJS, etc.	Some of the Nodejs modules are Lodash, express etc. These modules are to be imported from npm.
It is the upgraded version of ECMA script that uses Chrome's V8 engine written in C++.	Nodejs is written in C, C++ and Javascript.

# Why Node.js??

- Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications.
- Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.
- It makes use of event-loops via JavaScript's callback functionality to implement the non- blocking I/O.

- JavaScript used in client-side but node.js puts the JavaScript on server-side thus making communication between client and server will happen in same language
- Servers are normally thread based but Node.JS is “Event” based. Node.JS serves each request in a Evented loop that can able to handle simultaneous requests.

# Success Stories.....



Rails to Node

- « Servers were cut to 3 from 30 »
- « Running up to 20x faster in some scenarios »
- « Frontend and backend mobile teams could be combined [...] »



Java to Node

- « Built almost twice as fast with fewer people »
- « Double the requests per second »
- « 35% decrease in the average response time »

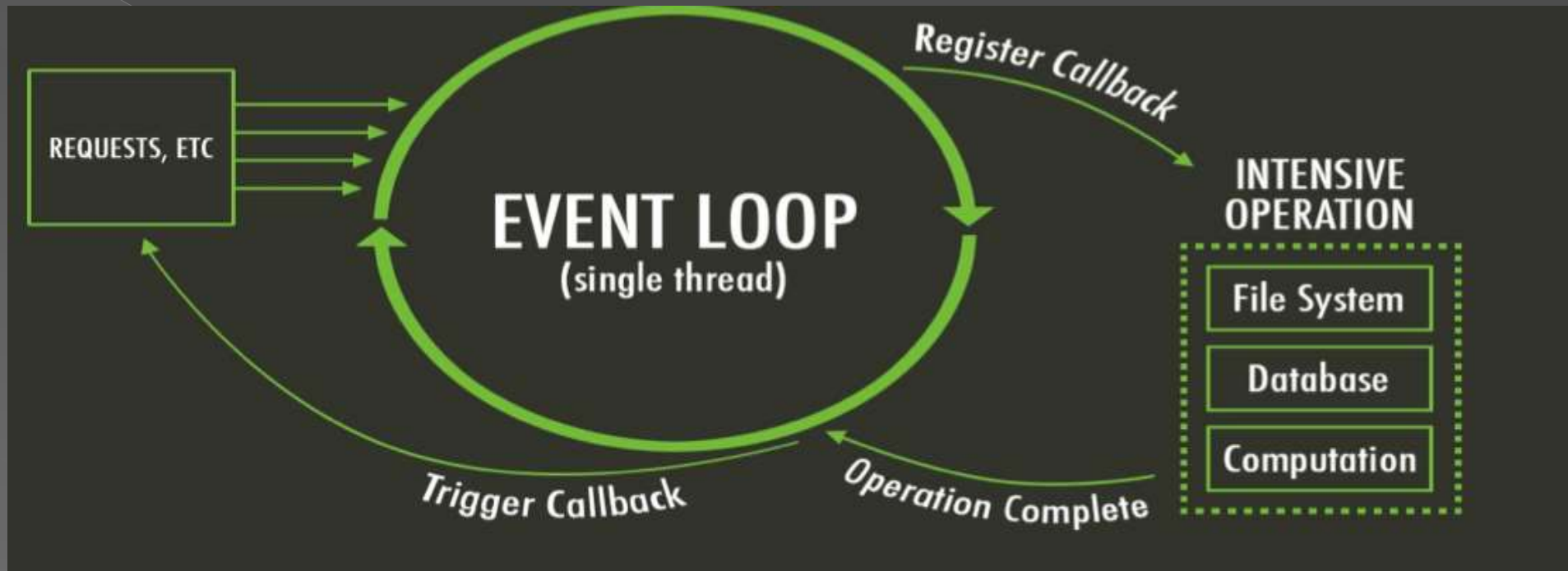
The Groupon logo, with the word "Groupon" in white on a black rectangular background.

GROUPON





# Event Loop



There are a couple of implications of this apparently very simple and basic model

- Avoid synchronous code at all costs because it blocks the event loop
- Which means: callbacks, callbacks, and more callbacks

# Why node.js use event-based?

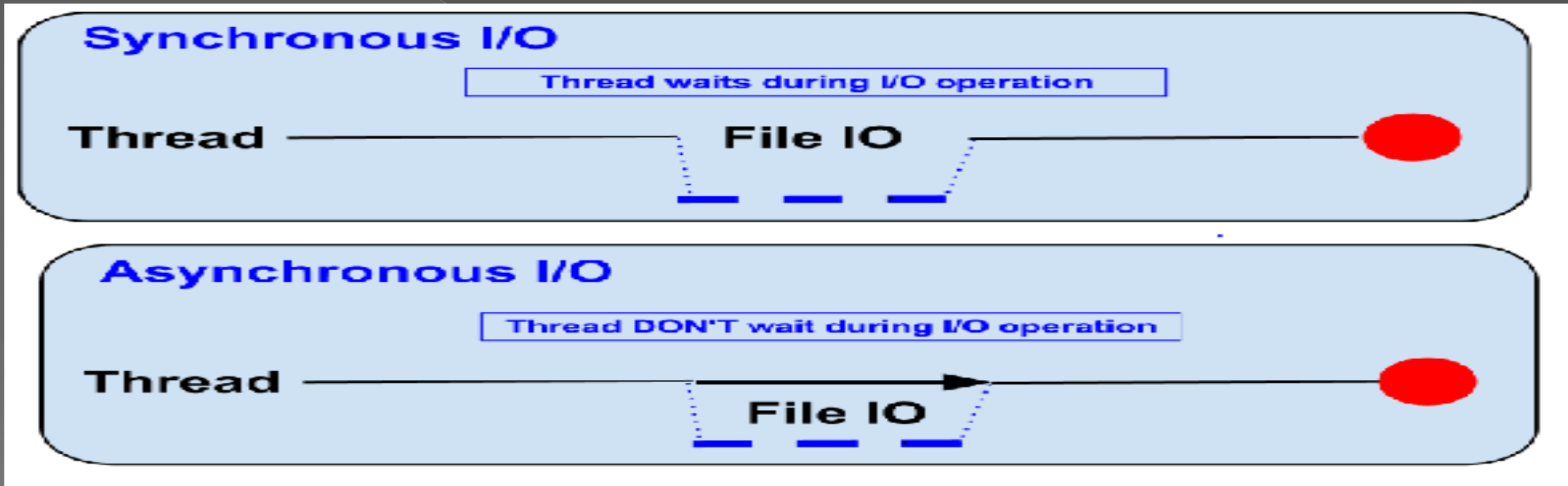
In a normal process cycle the webserver while processing the request will have to wait for the IO operations and thus blocking the next request to be processed.

Node.JS process each request as events, The server doesn't wait for the IO operation to complete while it can handle other request at the same time.

When the IO operation of first request is completed it will call-back the server to complete the request.

# Blocking vs Non-Blocking.....

Example : Read data from file and show data



# Blocking I/O

- Read data from file

- Show data

- Do other tasks

```
var data = fs.readFileSync( "test.txt" );  
console.log( data );  
console.log( "Do other tasks" );
```

# Non Blocking I/O



Callback

- Read data from file

When read data completed, show data

- Do other tasks

```
fs.readFile( "test.txt", function( err, data ) {  
  console.log(data);  
});
```

# What can we do with Node?

- We can create an **HTTP server** and print 'hello world' on the browser in just 4 lines of JavaScript.
- We can create a **DNS server**.
- We can create a **Static File Server**.
- We can create a **Web Chat Application** like GTalk in the browser.
- Node.js can also be used for creating **online games, collaboration tools** or anything which sends updates to the user in **real-time**.

# What can't do with Node?

- Node is a platform for writing JavaScript applications outside web browsers. This is not the JavaScript we are familiar with in web browsers. There is no DOM built into Node, nor any other browser capability.
- Node can't run on GUI, but run on terminal

# When to use Node.js

- Node.js is good for creating streaming based real-time services, web chat applications, static file servers etc.
- If you need high level concurrency and not worried about CPU-cycles.
- If you are great at writing JavaScript code because then you can use the same language at both the places: server-side and client-side.



# When not use Node.js

- Your server request is dependent on heavy CPU consuming algorithm/Job.
- Node.JS itself does not utilize all core of underlying system and it is single threaded by default, you have to write logic by your own to utilize multi core processor and make it multi threaded.

# Node.js Setup

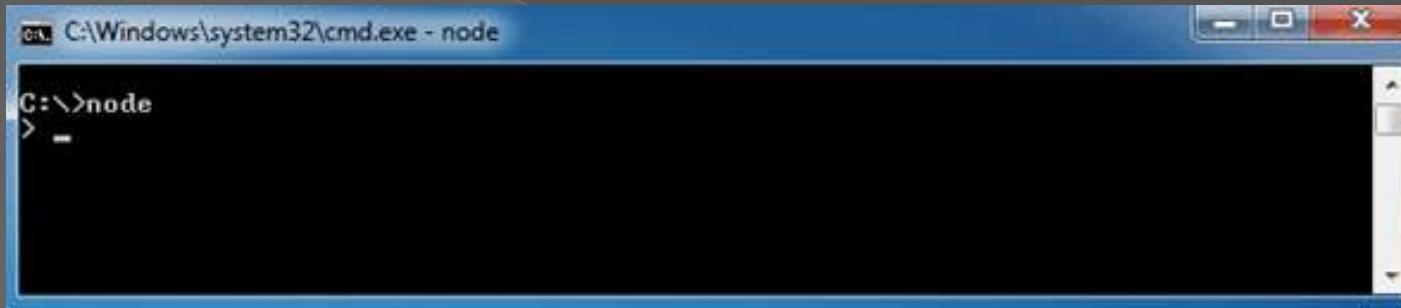
- The following tools/SDK are required for developing a Node.js application:
  - a. Node.js
  - b. Node Package Manager (NPM)
  - c. IDE (Integrated Development Environment) or Text Editor
- **Install Node.js on Windows:**
  - a. Visit Node.js official web site <https://nodejs.org>.
  - b. It will automatically detect OS and display download.
  - c. Download node MSI for windows by clicking on Download link.
  - d. After downloading the MSI, double-click on it to start the installation.
  - e. Click Next to read and accept the License Agreement and then click Install.
  - f. It will install Node.js quickly on your computer.
  - g. Finally, click finish to complete the installation
  - h. Open command prompt and type following.  
node -v  
npm -v

# Node.js Console - REPL

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks –

- Read – Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
- Eval – Takes and evaluates the data structure.
- Print – Prints the result.
- Loop – Loops the above command until the user presses ctrl-c twice.
- The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.

To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type node as shown below. It will change the prompt to > in Windows and MAC.



```
C:\Windows\system32\cmd.exe - node
C:\>node
>
```

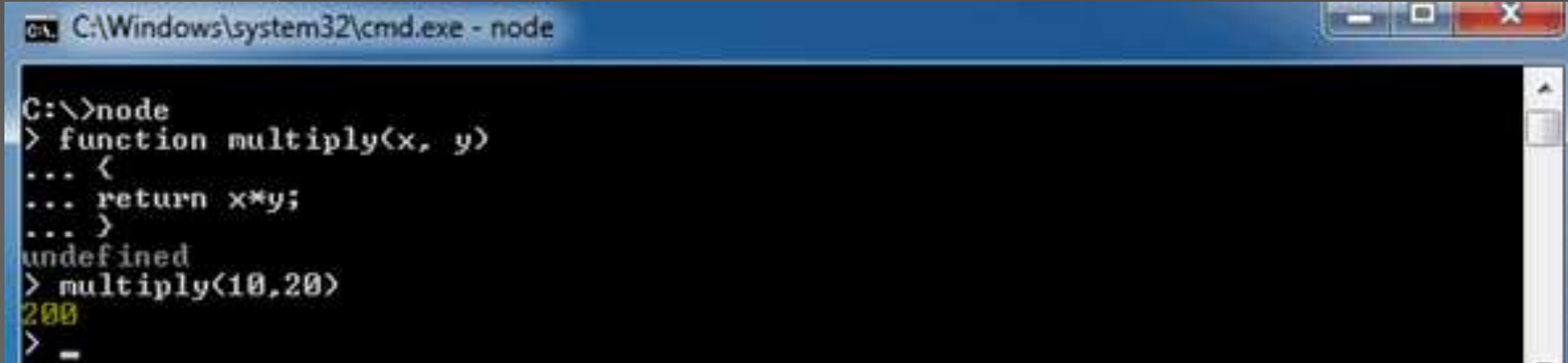
You can now test pretty much any Node.js/JavaScript expression in REPL. For example,

```
> 10 + 20
30
```

```
> "Hello " + "World"
Hello World
```

```
> var x = 10, y = 20;
> x + y
30
```

We can define a function and execute it as shown below.

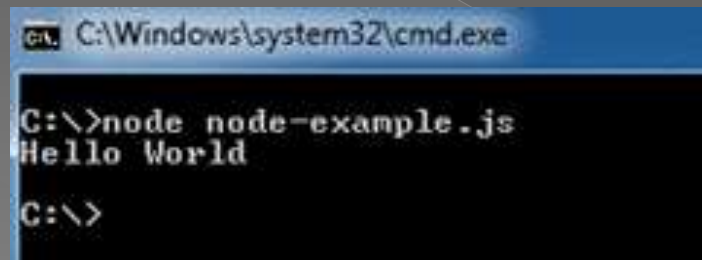


```
C:\>node
> function multiply(x, y)
... {
...   return x*y;
... }
undefined
> multiply(10,20)
200
> -
```

You can execute an external JavaScript file by writing node fileName command. For example,

node-example.js

```
console.log("Hello World");
```



```
C:\>node node-example.js
Hello World
C:\>-
```

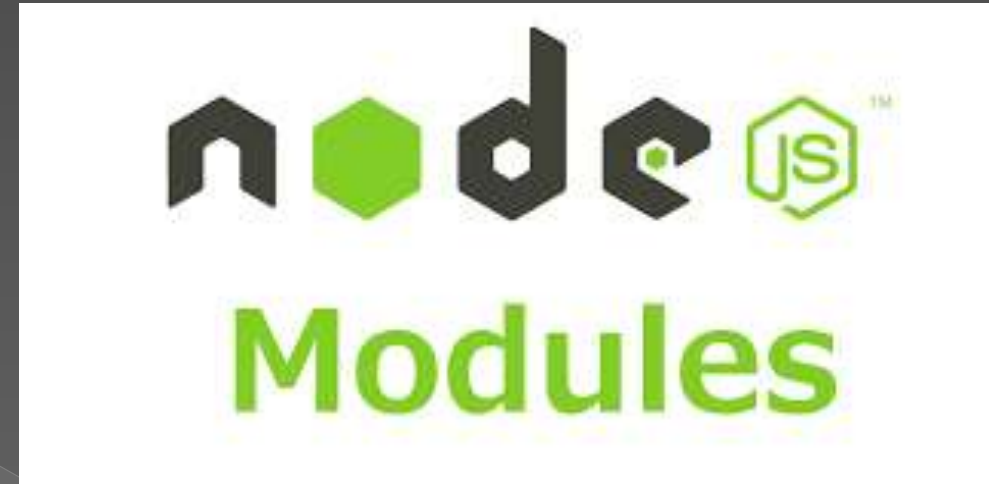
# REPL commands

The following table lists important REPL commands.

REPL Command	Description
<code>.help</code>	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
<code>.save filename</code>	Save current Node REPL session to a file.
<code>.load filename</code>	Load the specified file in the current Node REPL session.
<code>ctrl + c</code>	Terminate the current command.
<code>ctrl + c (twice)</code>	Exit from the REPL.
<code>ctrl + d</code>	Exit from the REPL.
<code>.break</code>	Exit from multiline expression.
<code>.clear</code>	Exit from multiline expression.

# Agenda for Today's Session

- Introduction to Node.js Modules
- Types of Module
- How to create a Local Module
- How to export module
- Require function
- Introduction to NPM
- Package.json
- NPM local Package
- NPM Global Package
- Adding dependency
- Update and Search Package
- Package-lock.json



# Modules

In Node, the modularity is a first-class concept. In the Node.js module system, each file is treated as a separate module. let's say, a demo.js file is a module.

Modules help us encapsulating our code into manageable chunks. Anything that we define in our module (i.e. in our JavaScript file) remains limited to that module only, unless we want to expose it to other parts of our code.



# Types of Modules

- **Core module:** Modules that come shipped with Node.js, e.g. **https**, **os**, **fs**, **net**, etc.
- **Third-party module:** Modules that you install from any package manager. We use these modules to accomplish or simplify any existing task. For example, to simplify our web API development we use **express**, or to deal with date and time we use **moment**.
- **Local module:** These are the modules that we create for our own use. These modules basically consist of core business logic of our code.

# Global Object

- `__dirname` :- It specifies the name of the directory that currently contains the code.
- `__filename` : - It specifies the filename of the code being executed.
- `setImmediate(callback[, arg][, ...])`
- `setInterval(callback, delay[, arg][, ...])`
- `setTimeout(callback, delay[, arg][, ...])`
- `clearImmediate(immediateObject)`
- `clearInterval(intervalObject)`
- `clearTimeout(timeoutObject)`

# How To Create a Module

**Creating a module in Node is very simple, just create a javascript file .**

Let's say we defined one file, named sum.js, with the following content:

```
const sum = (a, b) =>
{
    return a + b;
};
const result = sum(2, 3)
console.log(result)
```

# Exports

- This is an object used to expose our functionalities in one module, so these functionalities can be used in other modules.
- We can expose anything, this can be a function, variable, constants, classes, etc.
- whatever you assign to `module.exports` will be exposed as a module.

Message.js

```
module.exports = 'Hello world';
```

Log.js

```
module.exports.log = function (msg) {  
    console.log(msg);  
};
```

data.js

```
module.exports = {  
    firstName: 'James',  
    lastName: 'Bond'  
}
```

Log.js

```
module.exports = function (msg) {  
    console.log(msg);  
};
```

# Require

This is a function that we use to import or require the functionalities from other modules. It is a compliment to the exports object, which is used to export functionalities. require, on the other hand, is used to import those functionalities.

app.js

```
var msg = require('./Messages.js');  
  
console.log(msg);
```

app.js

```
var msg = require('./Log.js');  
  
msg.log('Hello World');
```

app.js

```
var person = require('./data.js');  
console.log(person.firstName + ' ' + person.lastName);
```

app.js

```
var msg = require('./Log.js');  
  
msg('Hello World');
```

# NPM

# Node Package Manager

- NPM is the world's largest Software Library (Registry).
- NPM is also a software Package Manager and Installer.
- The registry contains over 800,000 code packages.
- Open-source developers use npm to share software.
- Many organizations also use npm to manage private development.
- Created by Isaac Z. Schlueter

# Installing NPM

npm includes a CLI (Command Line Client) that can be used to download and install software:

```
C:\>npm install <package>
```

- npm is installed with Node.js
- All npm packages are defined in files called package.json.
- The content of package.json must be written in JSON.
- At least two fields must be present in the definition file: name and version



# Package.Json

Package.json holds various meta data relevant to the project. This file is used to give information to npm that allows it to identify project as well as handle the project's dependencies.

```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\administrator\Documents\NodeDemo>npm init --yes
Wrote to C:\Users\administrator\Documents\NodeDemo\package.json:

{
  "name": "NodeDemo",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}

C:\Users\administrator\Documents\NodeDemo>
```

# Attributes of Package.json

- name – name of the package
- version – version of the package
- description – description of the package
- homepage – homepage of the package
- author – author of the package
- contributors – name of the contributors to the package
- dependencies – list of dependencies. NPM automatically installs all the dependencies mentioned here in the node\_module folder of the package.
- repository – repository type and URL of the package
- main – entry point of the package
- keywords – keywords

# Installing Packages

There is a simple syntax to install any Node.js module –  
**syntax:**

```
$ npm install
```

For example, following is the command to install a famous Node.js web framework module called express

```
$ npm install express
```

We have two ways to install package.

1. Install package local
2. Install package global

# Install package local

Local packages are installed in the directory where you run `npm install`, and they are put in the `node_modules` folder under this directory.

By default, NPM installs any dependency in the local mode.

For Example: `$npm install express`

# Install package global

global packages are all put in a single place in your system (exactly where depends on your setup), regardless of where you run

```
npm install -g <package-name>
```

For example:

```
$ npm install express -g
```

# Adding dependency

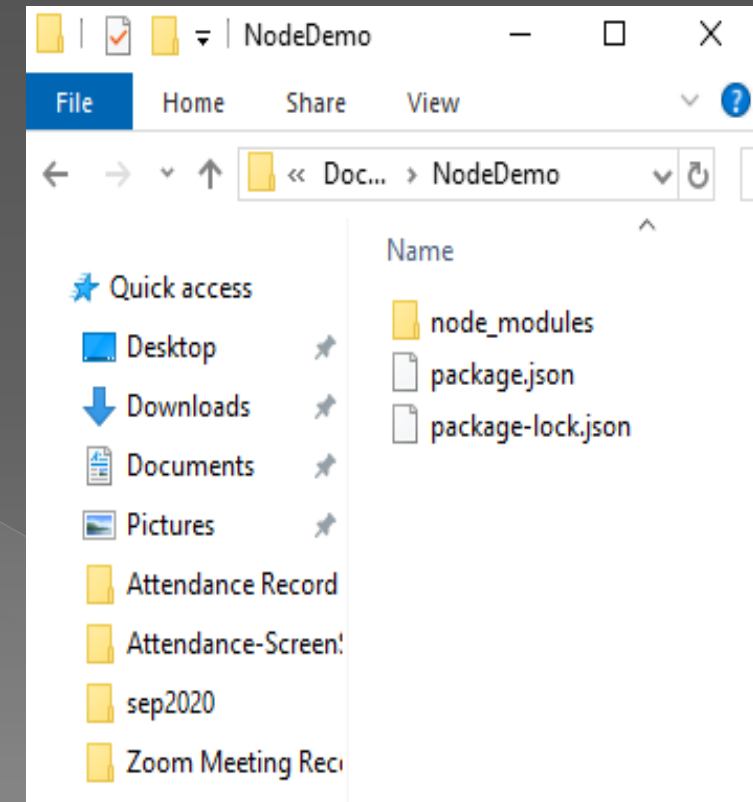
Adding dependency in package.json:  
For example:

```
Administrator: C:\Windows\System32\cmd.exe

C:\Users\administrator\Documents\NodeDemo>npm install express
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN NodeDemo@1.0.0 No description
npm WARN NodeDemo@1.0.0 No repository field.

+ express@4.17.1
added 50 packages from 37 contributors and audited 50 packages in 8.363s
found 0 vulnerabilities

C:\Users\administrator\Documents\NodeDemo>
```



# Adding dependency

```
{  
  "name": "NodeDemo",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo \"Error: no test specified\" && exit 1"  
  },  
  "keywords": [],  
  "author": "",  
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.17.1"  
  }  
}
```

# Updating packages

## ➤ Update Package:

Update package.json and change the version of the dependency to be updated and run the following command.

For example:

```
$ npm update express
```

## ➤ Search a package:

Search a package name using NPM.

For example:

```
$ npm search express
```



# package-lock.json

`package-lock.json` is automatically generated for any operations where npm modifies either the `node_modules` tree, or `package.json`. It describes the exact tree that was generated, such that subsequent installs are able to generate identical trees, regardless of intermediate dependency updates.

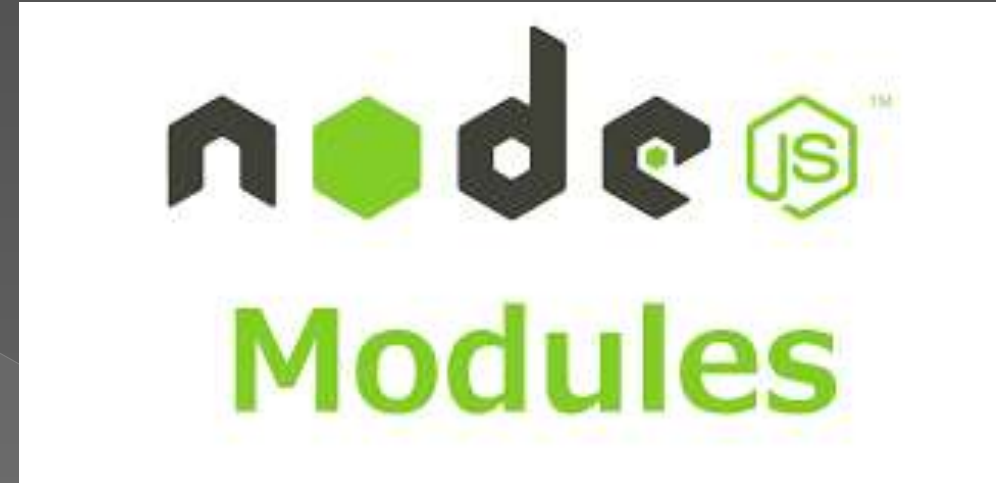
This file is intended to be committed into source repositories, and serves various purposes:

- Describe a single representation of a dependency tree such that teammates, deployments, and continuous integration are guaranteed to install exactly the same dependencies.
- Provide a facility for users to "time-travel" to previous states of `node_modules` without having to commit the directory itself.
- To facilitate greater visibility of tree changes through readable source control diffs.
- And optimize the installation process by allowing npm to skip repeated metadata resolutions for previously-installed packages.

# Agenda for Today's Session

## Core Modules-

- HTTP
- URL
- Fs
- Events



Node.js MySQL CRUD Operations

Node.js MongoDB CRUD Operations

# Core Modules

Node.js has a set of core modules that are part of the platform and come with the Node.js installation:

assert	buffer	child_process
console	cluster	crypto
dgram	dns	events
fs	http	http2
https	net	os
path	perf_hooks	querystring
readline	repl	stream
string_decoder	timers	tls
tty	url	util
v8	vm	wasi
worker	zlib	

# HTTP Module

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

Syntax:

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

**//create a server object:**

```
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('Hello'); //write a response to the client  
    res.end(); //end the response  
}).listen(1000); //the server object listens on port 1000
```

# URL Module

The URL module splits up a web address into readable parts.

```
var url = require('url');  
var adr = 'http://localhost:1000/default.htm?id=2&name=ram';  
var q = url.parse(adr, true);
```

```
console.log(q.host); //returns 'localhost:1000'  
console.log(q.pathname); //returns '/default.htm'  
console.log(q.search); //returns '?id=2&name=ram'
```

```
var qdata = q.query; //returns an object: { id: 2, name: 'ram' }  
console.log(qdata.name); //returns 'ram'
```

# fs Module

The File System module provides a way of working with the computer's file system.

Syntax:

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

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

# Reading File

## Reading File Asynchronously:-

```
var fs = require('fs');  
    fs.readFile('TestFile.txt', function (err, data)  
    {  
        if (err) throw err;  
        console.log(data);  
    });
```

## Reading File Synchronously:

```
var fs = require('fs');  
var data = fs.readFileSync('dummyfile.txt', 'utf8');  
console.log(data);
```

# Writing and Append Data File

## Creating & Writing File:-

```
var fs = require('fs');  
fs.writeFile('test.txt', 'Hello World!', function (err)  
{ if (err) console.log(err);  
else  
console.log('Write operation complete.');
```

## Append File Content:-

```
var fs = require('fs');  
fs.appendFile('test.txt', 'Hello World!', function (err) {  
if (err) console.log(err);  
else  
console.log('Append operation complete.');
```



# Delete File

```
var fs = require('fs');  
fs.unlink('test.txt', function () {  
  console.log('write operation complete.');
```

```
});
```

# Events

**Node.js allows us to create and handle custom events easily by using events module. Event module includes EventEmitter class which can be used to raise and handle custom events.**

```
// get the reference of EventEmitter class of events module  
var events = require('events');
```

```
//create an object of EventEmitter class by using above reference  
var em = new events.EventEmitter();
```

```
//Subscribe for FirstEvent  
em.on('FirstEvent', function (data) {  
  console.log('First subscriber: ' + data); });
```

```
// Raising FirstEvent  
em.emit('FirstEvent', 'This is my first Node.js event emitter example.');
```

# END

# Thank you !!