Node.js

📍What is Node.js:

- 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.
- It is open source and free to use. It can be downloaded from this link https://nodejs.org/en/
- Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

📍Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

- Extremely fast: Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
- I/O is Asynchronous and Event Driven: All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
- Single threaded: Node.js follows a single threaded model with event looping.
- Highly Scalable: Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
- No buffering: Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
- Open source: Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
- License: Node.js is released under the MIT license.

🏴Node.js First Example

Node.js console-based Example

File: console_example1.js

```
console.log('Hello JavaTpoint');
```

Node.js web-based Example

A node.js web application contains the following three parts:

Import required modules: The "require" directive is used to load a Node.js module.

Create server: You have to establish a server which will listen to client's request similar to Apache HTTP Server.

Read request and return response: Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.

```
var http = require("http");
http.createServer(function (request, response) {
 // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'});
   // Send the response body as "Hello World"
   response.end('Hello World\n');
}).listen(8081);
// Console will print the message
console.log('Server running at http://127.0.0.1:8081/');
```

✐Node.js Console

The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.

There are three console methods that are used to write any node.js stream:

- console.log()
- console.error()
- console.warn()

Node.js console.log()

The console.log() function is used to display simple message on console.

```
console.log('Hello JavaTpoint');
```

Node.js console.error()

The console.error() function is used to render error message on console.

```
console.error(new Error('Hell! This is a wrong method.'));
```

Node.js console.warn()

The console.warn() function is used to display warning message on console.

```
const name = 'John';
console.warn(`Don't mess with me ${name}! Don't mess with me!`)
```

Node.js REPL

The term REPL stands for Read Eval Print and Loop. It specifies a computer environment like a window console or a Unix/Linux shell where you can enter the commands and the system responds with an output in an interactive mode.

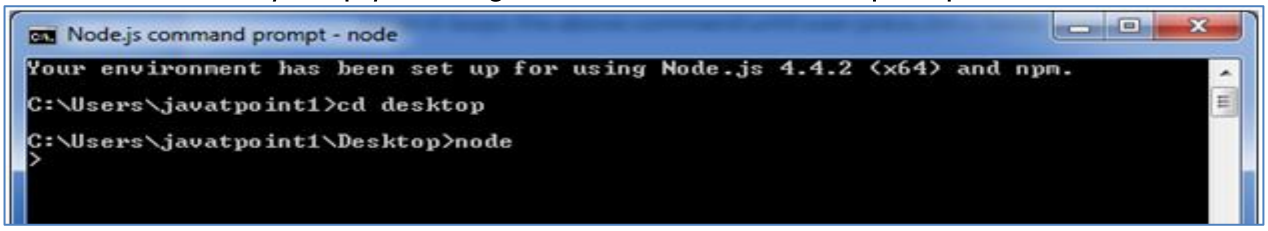
REPL Environment:

The Node.js or node come bundled with REPL environment. Each part of the REPL environment has a specific work.

- Read: It reads user's input; parse the input into JavaScript data-structure and stores in memory.
- Eval: It takes and evaluates the data structure.
- Print: It prints the result.
- Loop: It loops the above command until user press ctrl-c twice.

How to start REPL

You can start REPL by simply running "node" on the command prompt. See this:



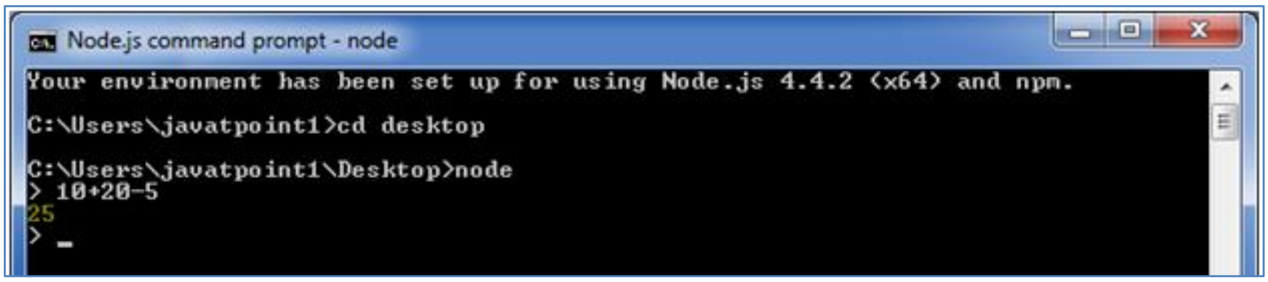
```
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.
C:\Users\javatpoint1>cd desktop
C:\Users\javatpoint1\Desktop>node
>
```

Node.js Simple expressions

After starting REPL node command prompt put any mathematical expression:

Example: >10+20-5

25



```
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.
C:\Users\javatpoint1>cd desktop
C:\Users\javatpoint1\Desktop>node
> 10+20-5
25
> _
```

🖉Node.js REPL Commands

Commands	Description
ctrl + c	It is used to terminate the current command.
ctrl + c twice	It terminates the node repl.
ctrl + d	It terminates the node repl.
up/down keys	It is used to see command history and modify previous commands.
tab keys	It specifies the list of current command.
.help	It specifies the list of all commands.
.break	It is used to exit from multi-line expressions.
.clear	It is used to exit from multi-line expressions.
.save filename	It saves current node repl session to a file.
.load filename	It is used to load file content in current node repl session.

🔷Node.js Package Manager(NPM):s

Node Package Manager provides two main functionalities:

- It provides online repositories for node.js packages/modules which are searchable on search.nodejs.org
- It also provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

The npm comes bundled with Node.js installables in versions after that v0.6.3. You can check the version by opening Node.js command prompt and typing the following command:

```
npm  version
```

Installing Modules using npm

Following is the syntax to install any Node.js module:

```
npm install <Module Name>
```

Let's install a famous Node.js web framework called express:

Open the Node.js command prompt and execute the following command:

```
npm install express
```

Global vs Local Installation:

1. npm install express -g

Uninstalling a Module

To uninstall a Node.js module, use the following command:

```
npm uninstall express
```

Searching a Module

"npm search express" command is used to search express or module.

```
npm search express
```

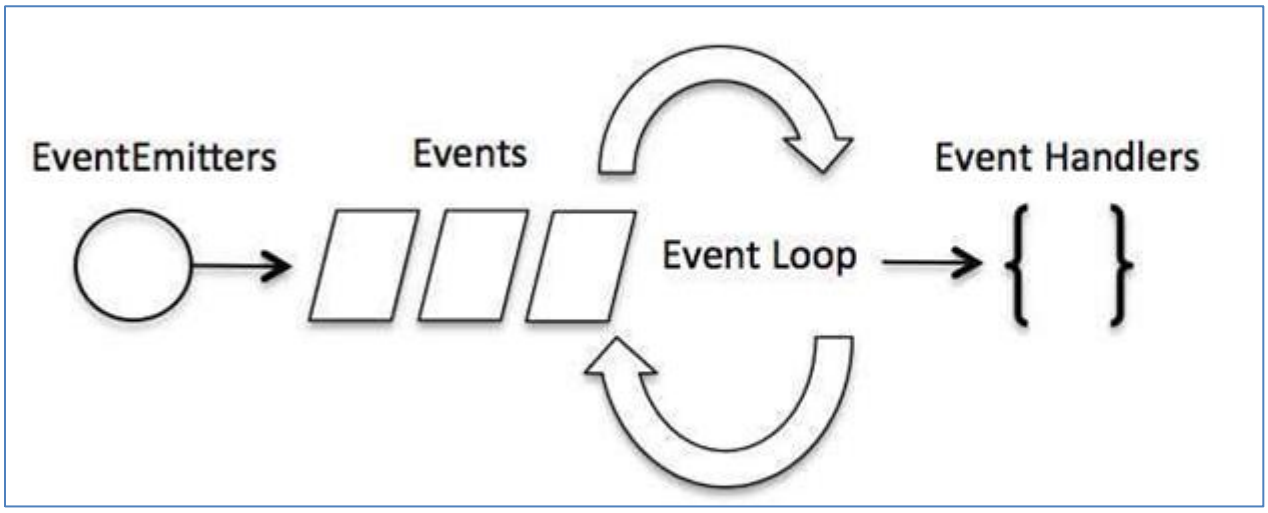
🔷Node.js Events:

In Node.js applications, Events and Callbacks concepts are used to provide concurrency. As Node.js applications are single threaded and every API of Node js are asynchronous. So it uses async function to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and after the completion of any task, it fires the corresponding event which signals the event listener function to get executed.

Event Driven Programming

Node.js uses event driven programming. It means as soon as Node starts its server, it simply initiates its variables, declares functions and then simply waits for event to occur. It is the one of the reason why Node.js is pretty fast compared to other similar technologies.

There is a main loop in the event driven application that listens for events, and then triggers a callback function when one of those events is detected.



To fire an event:

```
1.  // Fire an event
2.  eventEmitter.emit('eventName');
```

Node.js Event Example

File: main.js

```
// Import events module
var events = require('events');
// Create an eventEmitter object
var eventEmitter = new events.EventEmitter();

// Create an event handler as follows
var connectHandler = function connected() {
  console.log('connection succesful.');

  // Fire the data_received event
  eventEmitter.emit('data_received');
}

// Bind the connection event with the handler
eventEmitter.on('connection', connectHandler);
 // Bind the data_received event with the anonymous function
eventEmitter.on('data_received', function(){
  console.log('data received succesfully.');
});
// Fire the connection event
eventEmitter.emit('connection');
console.log("Program Ended.");
```

🔷Node.js Callbacks

Callback is an asynchronous equivalent for a function. It is called at the completion of each task. In Node.js, callbacks are generally used. All APIs of Node are written in a way to supports callbacks. For example: when a function start reading file, it returns the control to execution environment immediately so that the next instruction can be executed.

In Node.js, once file I/O is complete, it will call the callback function. So there is no blocking or wait for File I/O. This makes Node.js highly scalable, as it can process high number of request without waiting for any function to return result.

Blocking Code Example:

1. Create a text file named **input.txt** having the following content:

```
Javatpoint is an online platform providing self learning tutorials on
different technologies, in a very simple language.
```

2. Create a JavaScript file named **main.js** having the following code:

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

3. Open the **Node.js** command prompt and execute the following code.

```
node main.js
```

Non Blocking Code Example

Follow these steps:

1. Create a text file named **input.txt** having the following content:

```
Javatpoint is an online platform providing self learning tutorials on
different technologies, in a very simple language.
```

2. Create a JavaScript file named **main.js** having the following code:

```
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");
```

3. Open the **Node.js** command prompt and execute the following code.

```
node main.js
```

☑Node.js Buffers

Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap. Buffer class is used because pure JavaScript is not nice to binary data. So, when dealing with TCP streams or the file system, it's necessary to handle octet streams.

Buffer class is a global class. It can be accessed in application without importing buffer module.

Node.js Creating Buffers

There are many ways to construct a Node buffer. Following are the three mostly used methods:

1. **Create an uninitiated buffer:** Following is the syntax of creating an uninitiated buffer of 10 octets:

```
var buf = new Buffer(10);
```

2. **Create a buffer from array:** Following is the syntax to create a Buffer from a given array:

```
var buf = new Buffer([10, 20, 30, 40, 50]);
```

3. **Create a buffer from string:** Following is the syntax to create a Buffer from a given string and optionally encoding type:

```
var buf = new Buffer("Simply Easy Learning", "utf-8");
```

Node.js Writing to buffers

Following is the method to write into a Node buffer:

Syntax:

buf.write(string[, offset][, length][, encoding])

Parameter explanation:

- **string**: It specifies the string data to be written to buffer.
- **offset**: It specifies the index of the buffer to start writing at. Its default value is 0.
- **length**: It specifies the number of bytes to write. Defaults to buffer.length
- **encoding**: Encoding to use. 'utf8' is the default encoding.

Return values from writing buffers:

This method is used to return number of octets written. In the case of space shortage for buffer to fit the entire string, it will write a part of the string.

Let's take an example:

Create a JavaScript file named "main.js" having the following code:

File: main.js

```
buf = new Buffer(256);
len = buf.write("Simply Easy Learning");
console.log("Octets written : "+  len);
```

Node.js Streams

Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

- **Readable**: This stream is used for read operations.
- **Writable**: This stream is used for write operations.
- **Duplex**: This stream can be used for both read and write operations.
- **Transform**: It is type of duplex stream where the output is computed according to input.

Each type of stream is an Event emitter instance and throws several events at different times. Following are some commonly used events:

- **Data**: This event is fired when there is data available to read.
- **End**: This event is fired when there is no more data available to read.
- **Error**: This event is fired when there is any error receiving or writing data.
- **Finish**: This event is fired when all data has been flushed to underlying system.

(EX CHECK)

Networking module

Node.js Net

Node.js provides the ability to perform socket programming. We can create chat application or communicate client and server applications using socket programming in Node.js. The Node.js net module contains functions for creating both servers and clients.

Node.js Net Example

In this example, we are using two command prompts:

- Node.js command prompt for server.
- Window's default command prompt for client.

server:

File: net_server.js

```
const net = require('net');
var server = net.createServer((socket) => {
  socket.end('goodbye\n');
}).on('error', (err) => {
  // handle errors here
  throw err;
});
// grab a random port.
server.listen(() => {
  address = server.address();
  console.log('opened server on %j', address);
});
```

client:

File: net_client.js

```
const net = require('net');
const client = net.connect({port: 50302}, () => {//use same port of server
  console.log('connected to server!');
  client.write('world!\r\n');
});
client.on('data', (data) => {
  console.log(data.toString());
  client.end();
});
client.on('end', () => {
  console.log('disconnected from server');
});
```

What is Mean Stack?

Mean Stack refers to a collection of JavaScript technologies used to develop web applications. Therefore, from the client to the server and from server to database, everything is based on JavaScript.

MEAN is a user-friendly stack which is the ideal solution for building dynamic websites and applications. This free and open-source stack offers a quick and organized method for creating rapid prototypes for web-based applications.

MEAN is comprised of four different technologies:

- **MongoDB** express is a schemaless NoSQL database system
- **Express** JS is a framework used to build web applications in Node
- **AngularJS** is a JavaScript framework developed by Google
- **Node**.js is a server-side JavaScript execution environment

What is MongoDB?

MongoDB is an open-source, cross-platform database which is written in C++. It stores data in the key-value pair, using binary data type like JSON. It is a document-oriented NoSQL Database. A document in MongoDB resembles an Object in OOPS.

What is Express.JS?

Express is a mature, flexible, lightweight server framework. It is designed for building single, multi-page, and hybrid web applications. This lightweight framework uses the Pug engine to provide support for templates.

What is Angular JS?

Angular JS is an open-source JavaScript framework. Angular is maintained by Google. The goal of this framework is to introduce MVC(Model View Controller) architecture in the browser-based application that makes the development and testing process easier. The framework helps you create a smarter web app that supports personalization.

What is Node JS?

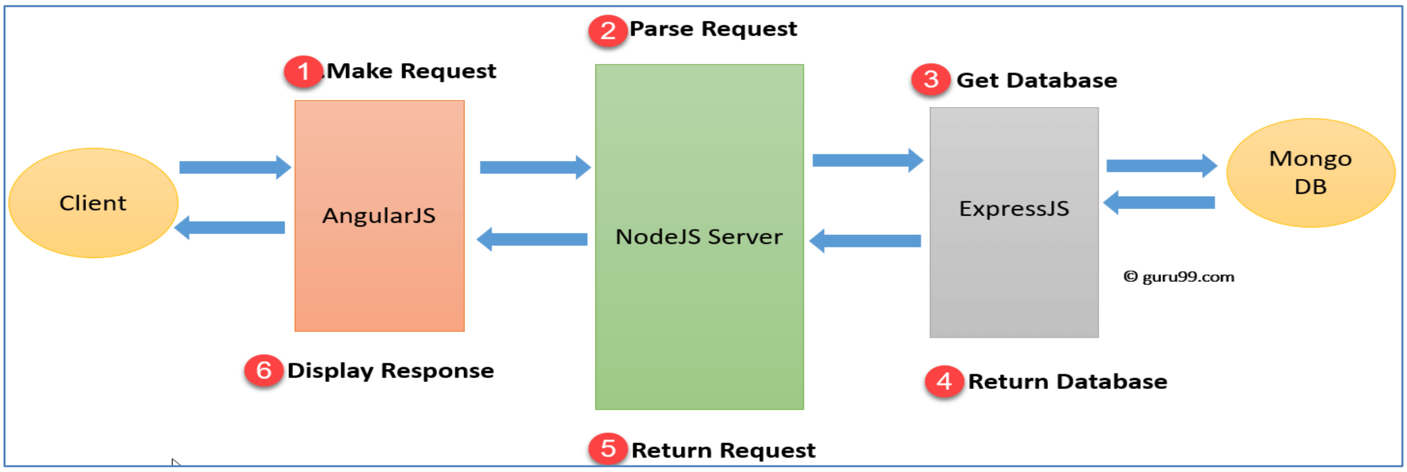
Node.js allows developers to create web servers and build web applications on it. It's a server-side Javascript execution environment.

Node.js uses a non-blocking and event-driven I/O model. This makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

Mean Stack Architecture

The primary function of various components of Mean Stack Architecture are as follows:

- Angular JS: Accept requests and display results to end user
- NodeJS: Handle Client and Server Requests
- Express JS: Make requests to Database and return a response
- MongoDB: Store and retrieve data.



1. Firstly, the client makes a request which is processed by the AngularJS
2. After that, the request moves to NodeJS which will parse the request.
3. ExpressJs will make calls to MongoDB to get or set data.
4. MongoDB will retrieve the requested data and return that request to the Express JS
5. NodeJS will return the request to the client.
6. At the client side, AngularJS to display the result fetched from MongoDB.

Advantages of Mean Stack

Here, are some most prominent reasons for using Mean Stack technology

- Allows creating a simple open source solution which can be used to build robust and maintainable solutions.
- Helps in rapid development of applications
- MEAN is full stack JavaScript which is 100% free. Leverage JavaScript's popularity
- Use a uniform language throughout your stack
- Uses very low memory footprint/overhead
- Helps you to avoid unnecessary groundwork and keeps your application organized
- MongoDB is built for the cloud
- Node.js simplifies the server layer
- MEAN makes code isomorphic

Disadvantages of Mean Stack

- MongoDB may be an ideal choice for small to the mid-sized application. However, it is not the best option for large-scale applications
- There are no specific general JS coding guidelines
- Once you have developed the first site using Mean stack technology, it's really hard to go back to the old approach
- It offers poor isolation of server from business logic
- You could potentially lose records