Complete notes on

Node JS

Written By:-
Dipti Gaydhane
(Software Engineer)

PAGE:
DATE: / /

INDEX

NODE.JS Tutorial

10) Node.js MySQL Create Database

11) Node.js MySQL Create Table

12) Node.js MySQL Select From

13) Node.js MySQL Where

14) Node.js MySQL Order By

15) Node.js MySQL Delete

16) Node.js MySQL Drop Table

17) Node.js MySQL Update

18) Node.js MySQL Limit

19) Node.js MySQL Join

20) Node.js MongoDB

21) Node.js MongoDB Create Database

22) Node.js MongoDB Create Collection

23) Node.js MongoDB insert

24) Node.js MongoDB Create Collection

25) Node.js MongoDB Insert

26) Node.js MongoDB Find

27) Node.js MongoDB Query

28) Node.js MongoDB Sort

29) Node.js MongoDB Delete

30) Node.js MongoDB Drop

31) Node.js MongoDB Update

32) Node.js MongoDB Limit

33) Node.js MongoDB Join

What is a Node.js File?

- Node.js files contain tasks that will be executed on certain events.
- A typical event is someone trying to access a port on the server.
- Node.js files must be initiated on the server before having any effect.
- Node.js files have extension ".js".

Node.js Get Started :

Download Node.js

The Official Node.js website has installation instructions for Node.js: https://nodejs.org

Getting Started

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a Web browser.

Create a Node.js file named "myfirst.js", and add the following code.

myfirst.js

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

http.createServer (function
(req, res) {
        res.WriteHead (200, {'content-
Type' : 'text/html'});
   res.end ('Hello World');
}).listen (8080);
```

Command Line Interface :

Node.js files must be initiated in the "Command Line Interface" program of your Computer.

```
C:\Users\your name>_
```

Initiate the Node.js File

The file you have just created must be initiated by Node.js before any action can take place.

Start your command line interface, write node myfirst.js and hit enter.

Initiate "myfirst.js":

```
C:\Users\Your Name>node myfirst.js
```

Node.js Modules :

What is a Module in Node.js ?

Consider modules to be the same as Javascript libraries.

A set of functions you want to include in your application.

Built-in Modules

Node.js has a set of built-in modules which

you can use without any further installation.

Includes Modules

To include a module, use the require() function with the name of the module.

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

Now your application has access to HTTP module, and is able to create a server.

```
http.createServer(function (req, res)
```

Create your Own Modules

You can create your own modules, and easily include them in your applications.

Example:

```
exports.myDateTime = function () {
    return Date();
};
```

Include your own Module

Now you can include and use the module in any of your Node.js files.

Example :-

Use the module "myfirstmodule" in a Node.js file.

```
Var http = require ('http');
Var dt =
require ('./myfirstmodule');
 res.write ("The date and time are
 currently: " + dt.myDateTime());
 res.end();
```

Node.js HTTP Module :

The Built-in 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)

Node.js as a Web Server

The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Use the `createServer()` Method to create in HTTP server

Example ↓

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

// create a server object:
http.createServer(function
(req, res) {
res.write('Hello World!');
// write a response to the client
 res.end(); // end the response
}).listen(8080);
// the server object listens on
port 8080
```

The function passed into the `http.createServer()` Method will be executed when someone tries to access the computer

On port 8080

Save the code above in file called "demo_http.js", and initiate the file:

Initiate demo_http.js:

```
C:\Users\your name> node demo_http.js
```

Add an HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type.

Example:

```
res.writeHead(200, {'Content-Type': 'text/html'});
res.write('Hello World!');
res.end();
```

Read the Query String

The function passed into the http.createServer() has a req argument that

represents the request from the client, as an object (http: IncomingMessage object).

```
res.writeHead ( 200 , { 'content-
Type' : 'text/ html' });
  res.write (req.url);
  res. end ();
```

```
C:\Users\your Name>node
demo_http_url.js
```

http://localhost:8080/summer

Result :

/ summer

http://localhost:8080/ winter

Result :

/ Winter

Split the Query String

Example → res.writeHead (200 , { 'content -

```
Type' : 'text/html' });
    var q = url.parse ( req.url,
true ).query ;
    var txt = q.year + " " + q.month ;
    res.end (txt);
```

The address :

http://localhost:8080/?
year = 2017&month = July

Result

2017 July

Node.js File System :

Node.js as a File Server

The Node.js file system module allows you to work with the file system on your computer.

To include file system module, use the require() Method.

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

Common Use for the File system Module :

- Read Files
- Create Files
- Update Files
- Delete Files
- Rename Files

Read Files

The fs.readFile() method is used to read files on your computer.

Assume we have the following HTML file (located in the same folder as Node.js):

demofile1.html

```
<html>
<body>
<h1> My Header </h1>
<p> My paragraph. </p>
</body>
</html>
```

Create a Node.js file that reads the HTML file, and return the content.

Example 1.

```
fs.readFile('demofile1.html',
            (err, data) {
    res.writeHead(200, {'Content-
Type': 'text/html'});
    res.write(data);
    return res.end();
});
```

Initiate demo_readfile.js:

```
C:\Users\your Name> node
demo_readfile.js
```

Creates File

The file system module has Methods for creating new files.

- fs.appendFile()
- fs.open()
- fs.writeFile()

ocean group

PAGE:
DATE: / /

fs.appendFile()

Example:

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

fs.appendFile('mynewfile1.txt',
'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved');
});
```

fs.open()

Example:

```
fs.open('mynewfile2.txt', 'w',
function (err, file) {
  (err) throw err;
  console.log('Saved');
});
```

fs.writeFile()

Example → `fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {`

```
  if (err) throw err;
  console.log ('Saved' );
});
```

Update Files

The file system module has methods for updating files.

- fs.appendFile ()
- fs. writefile ()

fs. appendFile ()

Example ↓

```
fs.appendfile ('mynewfile1.txt',
' This is my text.', function (err) {
    if (err) throw err;
  console.log ( 'updated!' );
});
```

fs. writefile ()

Example → fs.writefile ('mynewfile3.txt',
'This is my text', function (err) {

```
if (err) throw err;
console.log ('Replaced!');
});
```

Delete Files

fs.unlink()

Example ↓

```
fs.unlink ('mynewfile2.txt', function
(err) {
    if (err) throw err;
  console.log ('file deleted!');
});
```

Rename Files

fs.rename()

Example ↓

```
fs.rename ('mynewfile1.txt',
'myrenamedfile.txt', function (err)
```

```
{
  if (err) throw err;
  console.log('File Renamed!');
});
```

Node.js Send an Email :

The Nodemailer Module

The Nodemailer module makes it easy to send emails from your computer.

The Nodemailer module can be downloaded and installed using npm.

```
C:\Users\your Name>npm install
nodemailer
```

Send an Email

Now you are ready to send emails from your server.

Example → `var nodemailer = require('nodemailer');`

```
var transporter =
nodemailer.createTransport({
  service: 'gmail',
  auth: {
    user: 'youremail@gmail.com',
    pass: 'yourpassword'
  }
});

      mailoptions = {
  from: 'youremail@gmail.com',
  to: 'myfriend@yahoo.com',
  Subject: 'Sending Email using Node-js',
    text: 'That was easy!'
  };


transporter.sendmail(mailoptions,
  function (error, info) {
  if (error) {
  console.log(error);
} else {
    console.log('Email Sent: ' +
 info.response);
  }
});
```

Multiple Receivers

To send an email to more than one receiver, add them to the "to" property of the mailoptions object, seperated by commas:

Example ↓

Send email to More than one address.

```
var mailoptions = {
  from: 'youremail@gmail.com',
  to: 'myfriend@yahoo.com, myotherfriend@yahoo.com',
  Subject: 'Sending Email using Node.js',
  text: 'That was easy!'
}
```

Send HTML

To send HTML formatted text in your email, use the "html" property instead of the "text" property.

Example ↓

Send email containing HTML:

```
var mailoptions = {
  from: 'youremail@gmail.com',
  to: 'myfriend@yahoo.com',
  Subject: 'Sending Email using
  Node.js',
  html: '<h1>Welcome</h1><p> That
  Was easy!</p>'
}
```

Node.js MySQL*:

Node.js can be used in database applications.

One of the most popular database is MySQL.

MySQL Database

To be able to experiment with the code examples, you should have MySQL installed on your computer.

you can download a free MySQL database at https://www.mysql.com/downloads/.

Install MySQL Driver :

Once you have MySQL up and running on your computer, you can access it by using Node.js.

To access a MySQL database with Node.js, you need a MySQL driver. This tutorial will use the "mysql" module, downloaded from NPM.

To download and install the "mysql" Module, open the Command Terminal and execute the following.

```
C:\Users\Your Name>npm install mysql
```

Now you have downloaded and installed a mysql database driver.

Node.js can use this module to manipulate the MySQL database.

```
VAR    mysql = require ('mysql');
```

Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database.

demo_db_connection.js

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

  con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
```

Save the code above in a file called "demo_db_connection.js" and run the file.

Run "demo_db_connection.js"

```
C:\Users\your Name>node
demo_db_connection.js
```

Result

```
Connected!
```

Now you can start querying the database using SQL statements.

Querry a Database

Use SQL statements to read from (or write to) a Mysql database. This is also called "to query" the database.

The connection object created in the example above, has a Method for querying the database.

```
con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
```

```
con.query(sql, function
(err, result) {
    if (err) throw err;
    console.log("Result: " +
result);
  });
});
```

Node.js MySQL Create Database :

Creating Database

To create a database in MySQL, use the "CREATE DATABASE" statement.

Example ↓

```
var mysql = require(        );

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword"
});
```

```
con.connect(function(err) {
    if (err) throw err;
  console.log("Connected!");
  con.query("CREATE DATABASE mydb",
  function (err, result) {
     if (err) throw err;
     console.log("Database created");
  });
});
```

Save the code above in a file called "demo_create_db.js" and run the file

Run "demo_create_db.js"

```
C:\Users\Your Name>node
demo_create_db.js
```

Result

Connected!
Database created

Node.js MySQL Create Table :

Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection.

Example:

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

var con = mysql.createConnection({
  host: "localhost",
  user: "yourusername",
  password: "yourpassword",
  database: "mydb"
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
  var sql = "CREATE TABLE
customers (name VARCHAR(255),
```

```
address VARCHAR (255))";
    con.query (sql, function (err,
result) {
    if (err) throw
    console.log ("Table created");
  });
});
```

Save the code above in a file called "demo_create_table.js" and run the file.

Run "demo_create_table.js"

```
C:\Users\Your Name> node
demo_create_table.js
```

Result

```
Connected!
Table created
```

Insert Into Table

To fill the table in MySQL, use the "INSERT INTO" statement.

Example:

```
var mysql = require (          );

var con = mysql.createConnection ({
  host : "localhost",
  user : "yourusername",
  password : "yourpassword",
  database : "mydb"
});

con.connect (function (err) {
  if (err) throw err;
  console.log (
  var sql = " INSERT INTO
customers (name, address) VALUES
('Company Inc', 'Highway 37')";
  con.query (sql, function (err,
result) {
  if (err) throw err;
  (console.log (" 1 record
inserted");
  });
```

Save the code above in a file called "demo_db_insert.js", and run the file.

Run "demo_db_insert.js"

```
C:\Users\your name> node demo_db_insert.js
```

Result

```
Connected!
1 record inserted
```

Node.js MySQL Select From :

Selecting from a table

To select data from a table in MySQL, use the "SELECT" statement.

Example →
```
var mysql = require(        );
con.connect( function (err) {
if (err) throw err;
```

```
con.query("SELECT * FROM
customers", function (err, result,
fields) {
    if (err) throw err;
    console.log(result);
```

Node.js MySQL Where :

Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement.

Example :-

```
con.connect(function (err) {
  if (err) throw err;
  con.query("SELECT * FROM customers
WHERE address = 'park Lane 38'",
  function (err; result) {
    if (err) throw err;
    console.log (result);
  });
});
```

Node.js MySQL Order By :

Sort the Result

Use the ORDER By statement to sort the result in ascending or descending order.

The ORDER By keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

Example ↓

```
con.connect(function(err) {
  if (err) throw err;
  con.query("SELECT * FROM customers
ORDER By name", function (err,
result) {
    if (err) throw err;
    console.log(result);

  });
});
```

Node.js MySQL Delete :

Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

Example:

```
con.connect(function (err) {
  if (err) throw err;
  var sql = "DELETE FROM
customers WHERE address =
'Mountain 21'";
  con.query(sql, function(err,
result) {
    if (err) throw err;
  console.log("Number of records
  deleted: " + result.affectedRows);
  });
});
```

Node.js MySQL Drop Table :

Delete a Table

You can delete an existing table by using the "DROP TABLE" statement.

Example:

```
con.connect(function(err) {
if (err) throw err;
  sql = "DROP TABLE customers";
  con.query(sql, function(err, result) {
    if (err) throw err;
    console.log("Table deleted");
  });
});
```

Result

Table deleted

Node.js MySQL Update :

Update Table

Update existing records in a table by using

the "UPDATE" Statement.

Example ↓

```
con.connect ( function (err) {
if (err) throw err;
var sql = " UPDATE customers SET
address = 'Canyon 123' WHERE
Address = 'Valley 345' ";
con.query (sql, function (err,
result ) {
  if (err) throw err;
  Console . log ( result. affectedRows
+ " record(s) updated " );
});
});
```

Result

```
1 record(s) updated
```

Node.js MySQL Limit :

Limit the Result

limit the number of records returned

From the query, by using the "LIMIT" statement.

Example:

```
con.connect(function(err) {
  if (err) throw err;
var sql = "SELECT * FROM customers
LIMIT 5";
  con.query(sql, function(err,
result) {
    if (err) throw err;
    console.log(result);
  });
});
```

Node.js MySQL Join :

Join Two or More Tables

You can combine rows from two or more tables, based on a related column between them, by using a JOIN statement.

Consider you have a "users" table and a

"products" table.

Example:

```
con.connect(function(err) {
  if (err) throw err;
  var sql = "SELECT users.name AS
user, products.name AS favourite FROM
users JOIN products ON
users.favourite_product =
products.id";
    con.query(sql, function(err,
result) {
    if (err) throw err;
    console.log(result);
  });
});
```

Node.js MongoDB :

Node.js can be used in database applications.

One of the most popular NoSQL database is MongoDB.

MongoDB

You can download a free MongoDB database at https://www.mongodb.com.

Or get started right away with a MongoDB cloud service at https://www.mongodb.com/cloud/atlas.

Install MongoDB Driver

Let us try to access a MongoDB database with Node.js.

To download and install the official MongoDB driver, open the command Terminal and execute the following.

Dowload and install mongodb package.

```
C:\Users\Your Name> npm install mongodb
```

Node.js can use this module to manipulate MongoDB databases.

```
var Mongo = require('mongodb');
```

Node.js MongoDB Create Database :

Creating a Database

To create a database in MongoDB, start by creating a Mongoclient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

Example ↓

```
Var Mongoclient =
require ('morgodb').Mongoclient ;
var Url =
"mongodb://localhost:27017/mydb";

Mongoclient.connect(Url,
 function (err, db) {
  if (err) throw err;
 console.log("Database created!");
 db.close();
});
```

Run "demo_create_mongo_db.js"

```
C:\Users\your name> node
demo_create_mongo_db.js
```

Result

```
Database created!
```

Node.js MongoDB Create Collection :

A collection in MongoDB is the same as a table in MySQL.

Creating a Collection

To create a collection in MongoDB, use the createCollection() Method.

Example:

```
MongoClient.connect(url,
function(err, db) {
if (err) throw err;
var dbo = db.db("mydb");
```

```
dbo.createCollection (
function (err, res) {
 if (err) throw err;
 console.log (" Collection
created!");
 db.close ();
 });
});
```

Result

```
Collection created!
```

Node.js MongoDB Insert :

Insert Into Collection

To insert a record, or document as it is called in MongoDB, into a collection, we use the insertOne() Method.

A document in MongoDB is the same as a record in MySQL.

The first parameter of the insertOne() method is an object containing the name(s) and

value(s) each field in the document you want to insert.

Example:

```
MongoClient.connect(url,
function(err, db) {
 if (err) throw err;
 var dbo = db.db(      );
 var myobj = {name: "Company Inc",
address : "Highway 37"};

dbo.collection("customers").insertOn
e(myobj, function(err, res) {
    if (err) throw err;
   console.log("1 document
inserted");
    db.close();
  });
});
```

Result

```
1 document inserted
```

Node.js MongoDB Create Collection :

Creating a Collection

To create a collection in MongoDB, use the createCollection() Method.

Example:

```
MongoClient.connect(url,
 function(err, db) {
  if (err) throw err;
var dbo = db.db("mydb");
 dbo.createCollection("customers",
  function (err, res) {
   if (err) throw err;
   console.log("Collection
created!");
    db.close();
  });
});
```

Result

```
Collection created
```

Node.js MongoDB Insert :

Insert Into Collection

To insert a record, or document as it is called MongoDB, into a collection, we use the insertOne() Method.

A document in MongoDB is the same as a record in mySQL.

The first parameter of the Method is an object containing the name(s) and value(s) of each field in document you want to insert.

Example 1,

```
dbo.collection("customers").insertOne(myobj, function (err, res) {
    if (err) throw err;
    console.log("1 document inserted");
    db.close();
  });
});
```

Result

1 document inserted

Node.js MongoDB Find :

Find One

To select data from a collection in MongoDB, we can use the findOne() Method.

The findOne() Method returns the first Occurrence in the selection.

The first parameter of the findone() Method is a query Object. In this example we use an empty query Object, which selects all documents in a collection (but returns only the first document).

Example],

```
MongoClient.connect(url,
function(err, db) {
  if (err) throw err;
```

```
var dbo = db.db ("mydb");

dbo.collection ("customers").findOne(
{}, function (err, result) {
    if (err) throw err;
    console.log (result.name);
    db.close ();
  });
});
```

Node.js MongoDB Query :

Filter the Result

When finding documents in a collection, you can filter the result by using a query object.

The first argument of the find() method is a query object, and is used to limit the search.

Example :

```
MongoClient.connect (url,
```

```
function (err, db) {
  if (err) throw err;
var dbo = db.db ("mydb");
var query = { address : "Park
Lane 38" };

dbo.collection ("customers").find (query).
toArray ( function (err, result) {
    if (err) throw err;
    console.log (result);
    db.close ();
  });
});
```

Node.js MongoDB Sort :

Sort the result

Use the sort() method to sort the result in ascending or descending order.

Example↓

```
MongoClient.connect (url,
function (err, db) {
```

```
if (err) throw err;
var dbo = db.db("mydb");
var mysort = { name: 1 };

dbo.collection("customers").find().sort
(mysort).toArray( function (err,
result ) {
    if (err) throw err;
    console.log(result);
    db.close();
});
});
```

Node.js MongoDB Delete :

Delete Document

To delete a record, or document as it is called in MongoDB, we use the deleteOne() Method.

Example↓

```
MongoClient.connect(url,
function (err, db) {
```

```
if (err) throw err;
var dbo = db.db ("mydb");
var myquery = { address:
'Mountain 21' };

dbo.collection ("customers").deleteone
(myquery, function (err, obj) {
    if (err) throw err;
    console.log ("1 document
deleted");
    db.close ();
  });
});
```

Node.js MongoDB Drop :

Drop collection

You can delete a table, or collection as it is called in MongoDB, by using the drop() Method.

Example

```
dbo.collection ("customers").drop
```

```
( function (err, delok) {
  if (err) throw err;
  if (delok)
console.log ("collection deleted");
  db.close ();
  });
});
```

Result

```
collection deleted
```

Node.js MongoDB Update :

Update Document

You can update a record, or document as it is called in MongoDB, by using the `updateOne()` Method

The first parameter of the `updateOne()` Method is a query object defining which document to update.

Example → `MongoClient.connect(url,`

```
function (err, db) {
 if (err)        err;
 var dbo = db.db("mydb");
 var myquery = { address : "Valley 345" };
 dbo.collection ("customers") . updateone
 (myquery, newvalues, function (err, res ) {
        if (err) throw err;
        console.log ("1 document updated"
   db.close();
  });
});
```

Node.js MongoDB Limit :

Limit the Result

To limit the result in MongoDB, we use the limit() Method.

The limit() Method takes one parameter, a number defining how many documents to return.

Example ↓

```
dbo.collection ("customers") .find().limit
(5).toArray(function (err, result)
```

```
    {
  if (err) throw err;
    console.log(result);
    db.close();
  });
});
```

Node.js MongoDB Join :

MongoDB is not a relational database, but you can perform a left outer join by using the $lookup stage.

The $lookup stage lets you specify which collection you want to join with the current collection, and which fields that should match.

Example :

```
dbo.collection('orders').aggregate(
[ { $lookup :
      {
        from : 'products',
        localField : 'product_id',
```

```
            foreignfield: '_id',
            as: 'orderdetails'
        }
    }
]).toArray(function (err, res) {
    if (err) throw err;

    console.log(JSON.stringify(res));
    db.close();
});
});
```

Result

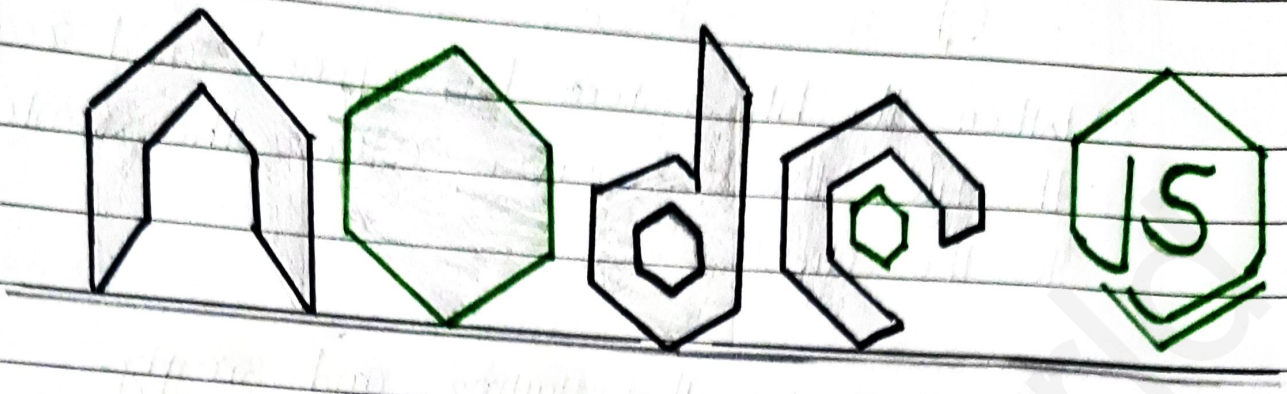
```
[
  { "_id": 1, "product_id": 154,
"status": 1, "orderdetails": [
    { "_id": 154, "name":
"Chocolate Heaven" }]
  }
]
```

Example↓

```
MongoClient.connect(url,
function (err, db) {
```

```
    if (err) -      err;
    var dbo = db.db ( "mydb" );
    var myquery = { address : " valley
345 " };
        newvalues = { $set : { name:
" Mickey, address : " Canyon 123" } };

dbo. collection ( " customers") . updateone
(myquery , newvalues, function (err,
 res) {
    if (err) throw err;
    Console .log ( " 1 document
updated" );
    db. close ();
  });
});
```



Node.js Introduction :

What is Node.js ?

- Node.js is an open source server environment
- Node.js is Free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses Javascript on the server

Why Node.js ?

1. Sends the task to the computer's File system.