a	Open file for appending. the file is created if It does not exist.
ax	like 'a' but fails if path exists.
a+	open file for reading and appending. the file is created if it does not exist.
ax+	Open file for reading and appending. the file is created if it does not exist.

Create a JavaScript file named "main.js" having the following code to open a file input.txt for reading and Writing.

File: main.js

```
Var fs = require("fs");
//Asynchronous - Opening File
Console.log("Going to open file!");
fs.open('input.txt', 'r+' function(err, fd) {
    if (err){
        return console.error(err);
    }
    console.log("File opened sucessfully!");
});
```

Node.js File Information Method

fs.Stat (path, callback)

Node.js fs. Stats Class Methods

Stats.isfile()

returns true if file type of a Simple file.

Stats.isdirectory()
returns true if file type of a directory.

Stats.isblockdevice()
returns true if file type of a block device.

Stats.ischaracterdevice()
returns true if file type of a character device.

Stats.issymboliclink()
returns true if file type of a symbolic link.

Stats.isfifo()
returns true if file type of a fifo.

Stats.issocket()
returns true if file type of asocket.

Lets take an example to create a JavaScript file named main.js having the following code:

File: main.js

```
Var fs = require ("fs");
 console.log ('Going to get file info!");
 fs.stat ('input.txt' function (err,Stats){
    if (err) {
       return console.error(err);
    }
    Console.log (stats);
    console.log ("Got file Info successfully!");
    // check file type
    console.log ("isfile?" + stats.isFile());
    console.log("is Directory?" +status.isDirectory()
});
```

Node.js Path

The Node.js path module is used to handle and transform files paths. This module can be imported by using the following syntax:

```
Var path = require("path")
```

Node.js Path Methods

1. path.normalize(p)
It is used to normalize a string path, taking care of '..' and '.' parts.

2. path.join([path1][,path2][,.....])
It is used to join all arguments together and normalize the resulting path.

3. Path.resolve([from ...], to)
It is used to resolve an absolute path.

4. path.isabolute(path)
It determines Whether path is an absolute path. an absoulte path Will always resolve the same location, regardless of the Working directory.

5. path.relative(from,to)
It is used to solve the relative path from "from" to "to".

6. path.dirname(p)
It is used to solve the relative it returns the directory name of a path. It is similar to the Unix base-dir-

name command.

7. path.basename([p[, ext]])
It returns the last portion of a path. It is similar to the Unix basename command.

8. path.extname(p)
It returns the extension of the path, from the last '.' to end of string in the last portion of the path. if there is no '.' in the last portion of the parth or the first character of it is '.', then it returns an empty string.

9. path.parse(pathstring)
It returns an object from a path string.

10. path.format(pathobject)
It returns a path string from an object, the opposite of path.parse above.

Node.js Path Example

File: path_example.js

```
Var path = require ("path");
// Normalization
Console.log('normalization:' + normalize('/ssit/javatpoint
            //node/newfolder/tab/..'));
// Join
console.log ('joint path: ' + path.join('/sssit', 'javatpoint,
            'node/newfolder', 'tab', '..'));
// Resolve
Console.log ('resolve: ' + path.resolve('path_example.js')
//Extension
```

```
console.log('ext name: ' + path.extname('path_example.js'));
```

Node.js String.Decoder

The Node.js String.Decoder is used to decode buffer into string. It is similar to buffer.toString() but provides extra support to UTF.

You need to use require('string_decoder').StringDecoder;

Node.js StringDecoder Methods.

StringDecoder class has two methods only.

- **decoder.write(buffer)**

  It is used to return the decoded string.

- **decoder.end()**

  It is used to return trailing bytes, if any left in the buffer.

Node.js StringDecoder Example

Let's see a simple example of Node.js StringDecoder.

File: stringDecoder_example1.js

```
Const StringDecoder = require('string_decoder').StringDecoder
const decoder = new StringDecoder('utf8');
const buf1 = new Buffer('this is a test');
console.log(decoder.write(buf1));//prints: this is a test
const buf2 = new Buffer('7486973206973206120746C3a97374',
                        'hex');
console.log(decoder.write(buf2);//prints: this is a test
Const buf3 = Buffer.from([0x62,0x75,0x66,0x66,0x65,0x72]);
console.log(decoder.write(buf3));//prints: buffer
```

Node.js Query String

The Node.js Query String provides methods to deal With query string. It can be used to convert query string into JSON object and vice-versa.
To use query string module, you need to use require(querystring)

Node.js Query String Methods.

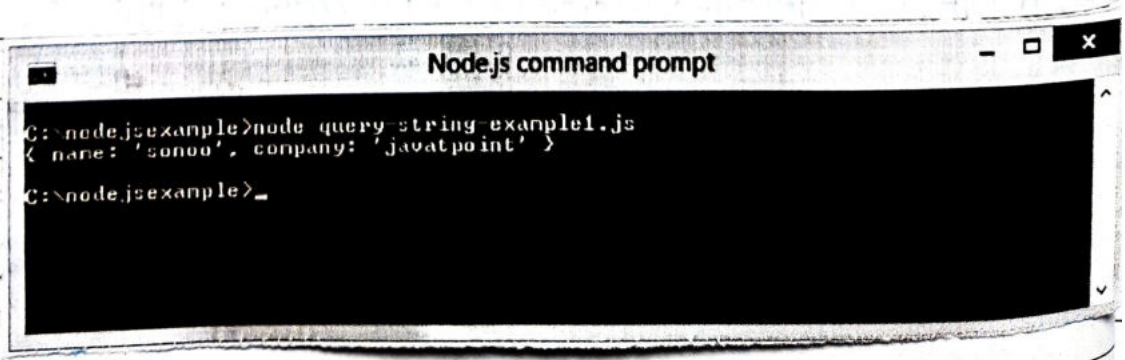
1. querystring.parse(str[,sep][,eq][,options])
Converts query string into JSON object.

2. querystring.stringify(obj[,sep][,eq][,options])
Converts JSON object into query string.

Node.js Query String Example 1: parse()

File: query_string_example-1.js

```
Querystring = require('querystring')
const obj1 = querystring.parse('name=sonoo&
                company=java tpoint');
console.log(obj1);
```



Node.js ZLIB

The Node.js zlib module is used to provide compression and decompression (zip and unzip) functionalities. It is implemented using Gzip and deflate/inflate.
The zlib module can be accessed using:

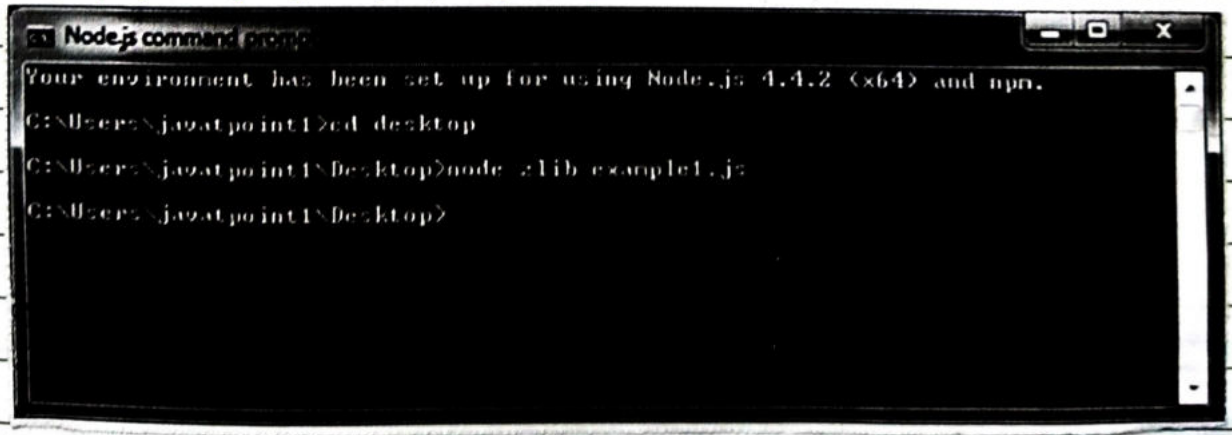
```
Const zlib = require('zlib');
```

Compressing and decompressing a file can be done by piping the source stream data into a destination stream through zlib stream.

Node.js ZLIB Example : Compress File

File: zlib_example1.js

```
Const zlib = require('zlib');
Constgzip = zlib.createGzip();
Const fs = require('fs');
Const inp = fs.createReadStream('input txt');
Const out = fs.createWritestream('input txt.gz');
inp.pipe(gzip).pipe(out)
```



Node.js ZLIB Example : Decompress File

File: zlib_example2.js

```
Const zlib = require('zlib');
Const unzip = zlib.createUnzip();
```

```
Const fs = require('fs');
Const inp = fs.createReadStream('input.txt.gz');
Const out = fs.createWritestream('input2.txt');
inp.pipe(unzip).pipe(out);
```

Node.js Assertion Testing

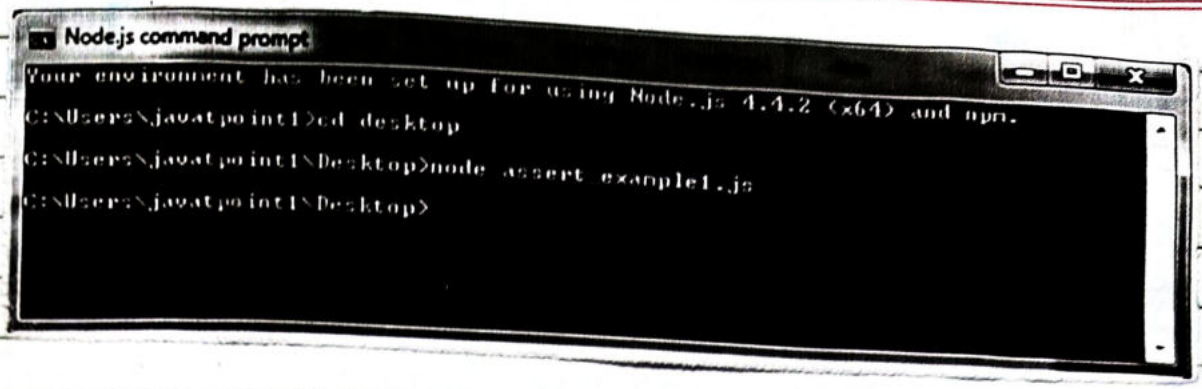
The Node.js Assert is the most elementary way to Write tests. It provides no feedback When running your test unless one fails. The assert module provides a simple set of assertion tests that can be used to test invariants. The module is intended for internal use by Node.js, but can be used in application code via require ('assert'). However assert is not a testing framework and cannot be used as general purpose assertion library.

Node.js Assert Example

File: assert_example1.js

```
Var assert = require('assert');
function add (a,b){
   return a+b;
}
Varexpected = add(1,2);
assert(expected===3,'One plus two is three');
```

It Will not provide any output because the case is true. If you Want to See output, you need to make the test fail.



Node.js V8

What is V8

V8 is an open source JavaScript engine developed by the chromium project for the Google Chrome Web browser. It is written in C++. Now a days, it is used in many projects such as couchbase, mongoDB and Node.js.

V8 in Node.js

The Node.js V8 module represents interfaces and event specific to the version of V8. It provides methods to get information about heap memory through V8.getHeapStatistics() and V8.getHeapSpacestatistics() methods. To use this module, you need to use require('v8').

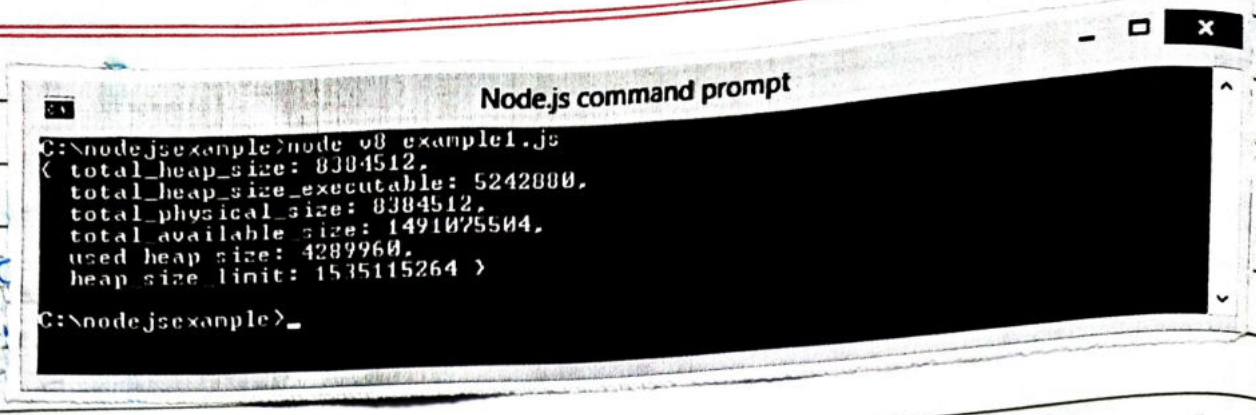
```
Const V8 = require('V8');
```

Node.js V8.getHeapStatistics() Example

The V8.getHeapStatistics() method returns statistics about heap such as total heap size, used heap size, heap size limit, total available size etc.

File: V8_example1.js

```
Const V8 = require('V8');
console.log (V8.getHeapStatistics());
```



Node.js V8.getHeapSpaceStatistics() Example

The V8 getHeapSpaceStatistics() returns statistics about heap space. It returns an array of 5 objects: new space, old space, code space, map space and large object space. Each object contains information about space name, space size, space used size, space available size and physical space size.

File: V8_example2.js

```
Const V8 = require('V8');
Console.log (V8.getHeapSpaceStatistics());
```

```
C:\nodejsexample>node v8-example2.js
[ { space_name: 'new_space',
    space_size: 2097152,
    space_used_size: 1021376,
    space_available_size: 10560,
    physical_space_size: 2097152 },
  { space_name: 'old_space',
    space_size: 3063808,
    space_used_size: 2465304,
    space_available_size: 0,
    physical_space_size: 3063808 },
  { space_name: 'code_space',
    space_size: 2097152,
    space_used_size: 613152,
    space_available_size: 2304,
    physical_space_size: 2097152 },
  { space_name: 'map_space',
    space_size: 1126400,
    space_used_size: 191488,
    space_available_size: 0,
    physical_space_size: 1126400 },
  { space_name: 'large_object_space',
    space_size: 0,
    space_used_size: 0,
    space_available_size: 1491062528,
    physical_space_size: 0 } ]
C:\nodejsexample>
```

Memory limit of V8 in Node.js

Currently, by default V8 has a memory limit of 512mb on 32-bit and 1gb on 64-bit systems. You can raise the limit by setting --max-old-space-size to a maximum of ~1gb for 32-bit and ~1.7gb for 64-bit systems. But it is recommended to split your single process into several Workers if you are hitting memory limits.

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.

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.

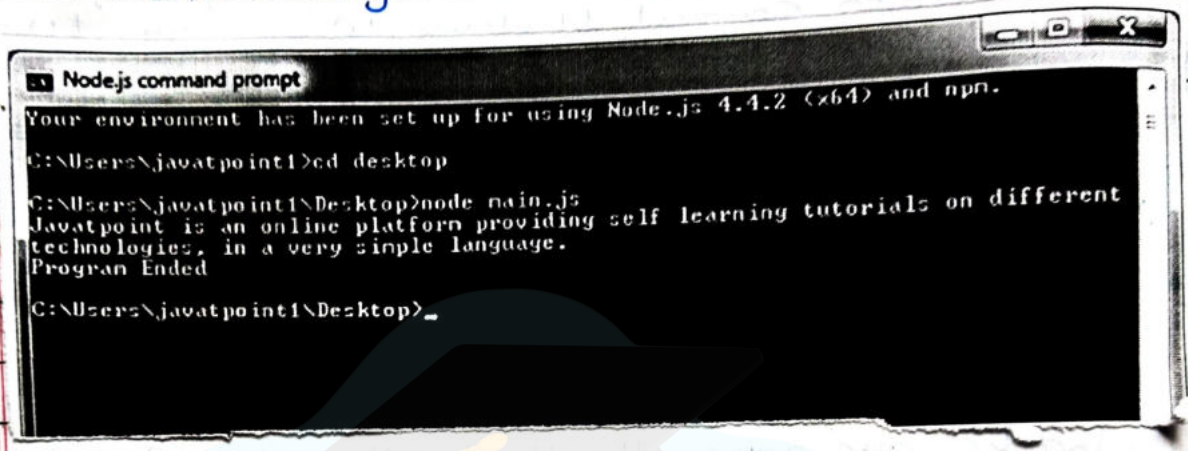
2. Create a JavaScript file named main.js having the following code:

```
Var fs = require("fs");
Var data = fs.readFileSync('input.txt');
```

```
Console.log (data. toString());
Console.log ("program Ended ");
```

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

```
Ndde main.js
```



Non Blocking Code Example

Follow these steps:

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

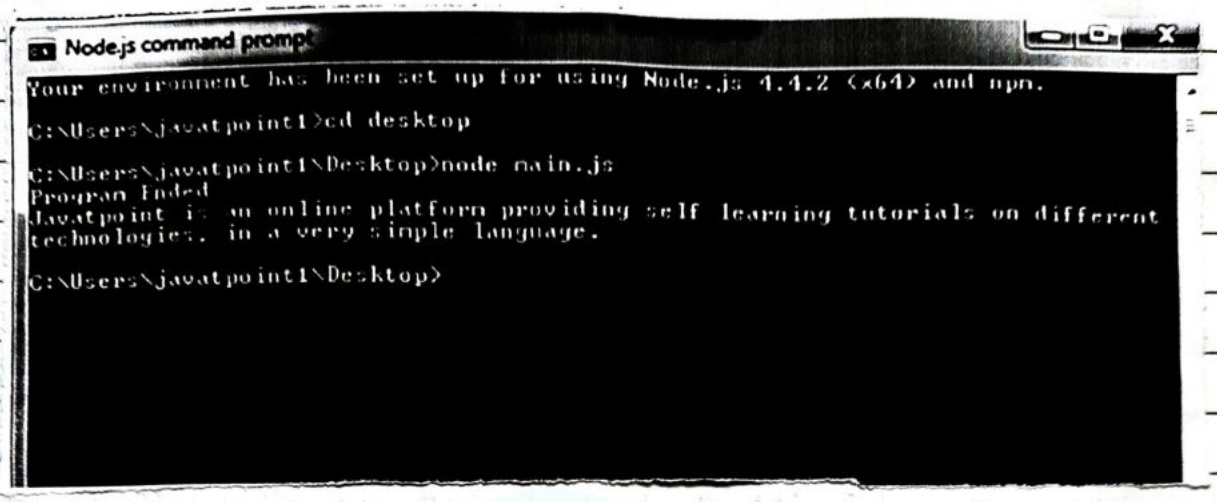
Javatpoint is an online platform providing self learning tutorials on different technologies, in a very simple language.

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

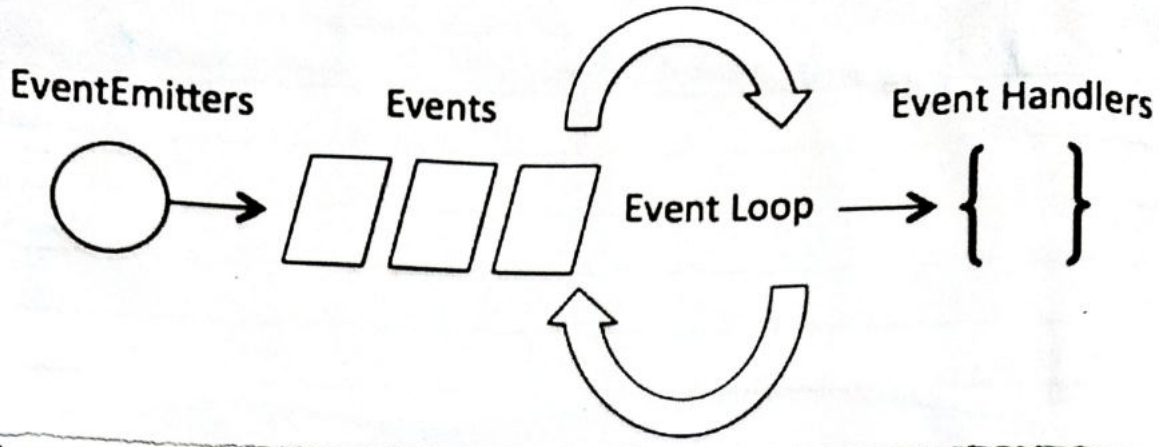


Node.js Events

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 observe 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 intiates its variables, declares function and then simply waits for event to occur. It is the one of the reson why Node.js is preety 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.



EventEmitter class to bind event and event listener:

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

To bind event handler with an event:

```
//Bind event and even handler as follows
eventEmitter.on('eventName', eventHandler);
```

To fire an event:

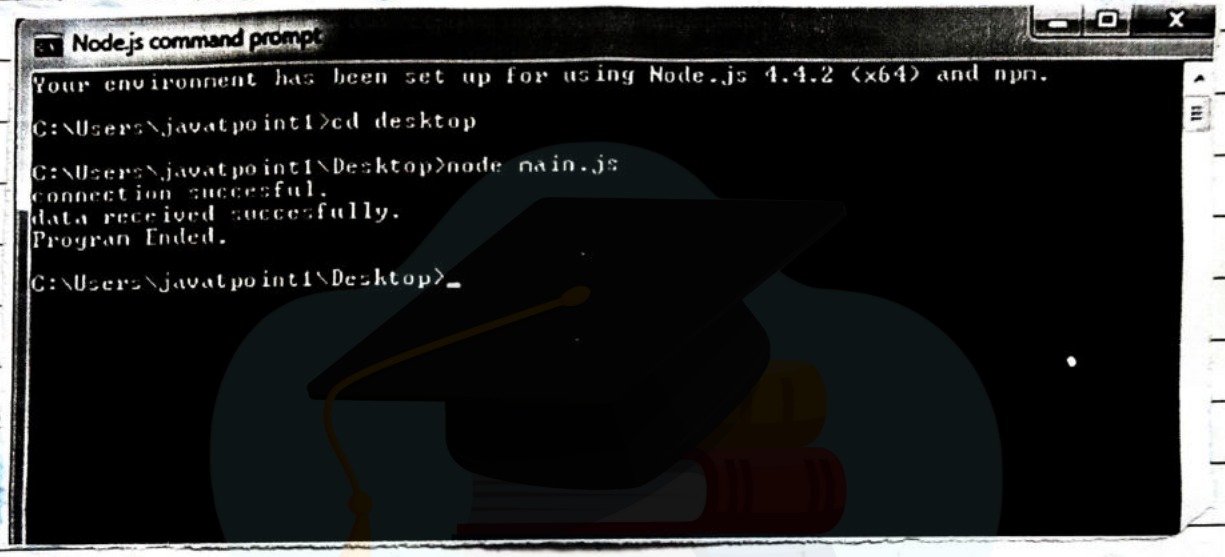
```
// Fire an event
eventEmitter.emit('eventName');
```

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 funct
eventEmitter.on('data_received', function() {
    Console.log('data received succesfully.');
});
// Fire the connection event
 eventEmitter.emit('connection');
 Console.log("program Ended.");
```



Node.js Punycode

What is Punycode

Punycode is an encoding syntax Which is used to convert Unicode (UTF-8) string of characters to basic ASCII string of characters. Since host names only understand ASCII characters so punycode is used. It is used as an Internationalizd domain name (IDN or IDNA)

Punycode in Node.js

Punycode.js is bundled with Node.js v0.6.2 and later version. If you want to use it With other Node.js versions then use npm to install punycode module first. You have to use require('punycode') to access it.

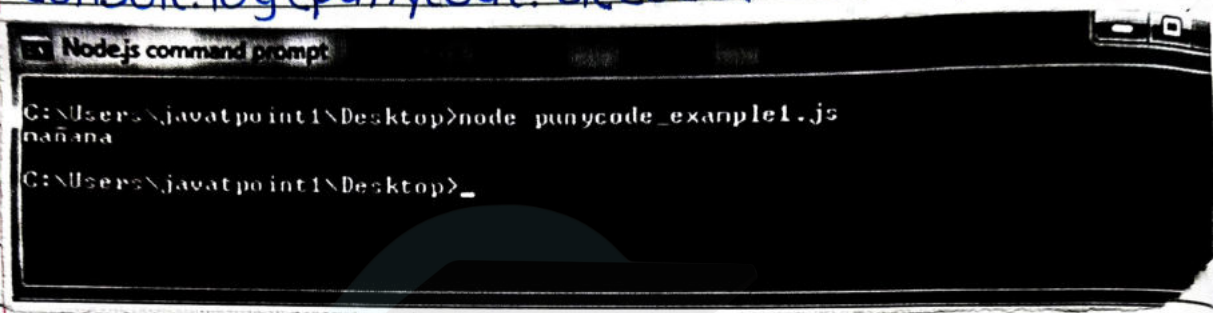
```
punycode = require('punycode');
```

punycode.decode(string)

It is used to convert a Punycode string of ASCII symbols to a string of Unicode symbols.

File: punycode_example1.js

```
punycode = require('punycode');
console.log(punycode.decode('maana-pta'));
```

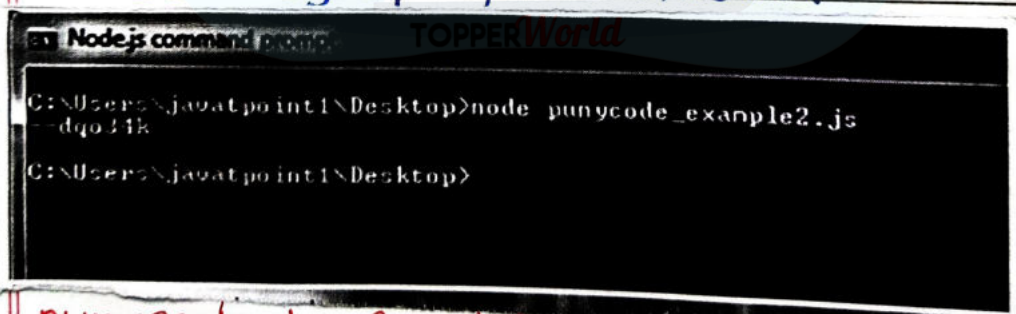


punycode.encode(string)

It is used to convert a string of Unicode symbols to a punycode string of ASCII symbols.

File: punycode_example2.js

```
punycode = require('punycode');
console.log(punycode.encode('
```

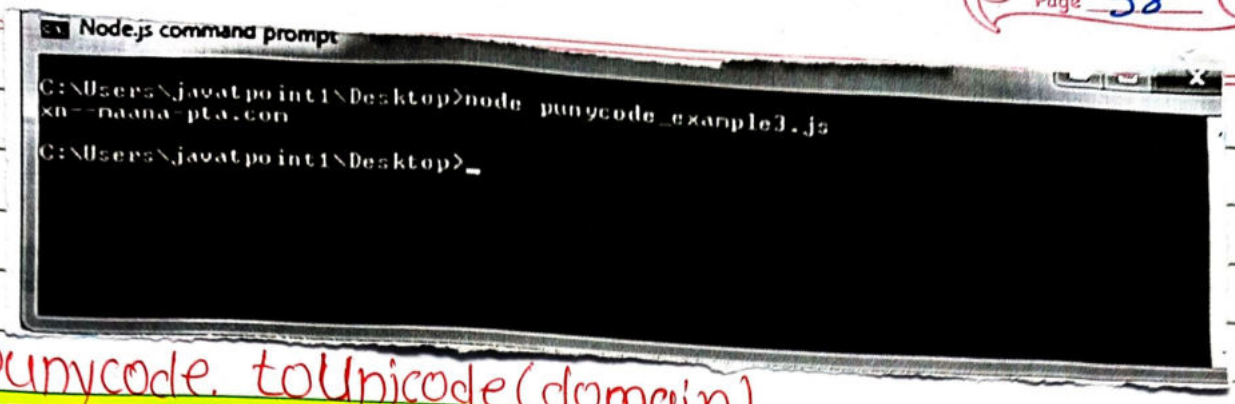


punycode.toASCII(domain)

It is used to convert a Unicode String representing a domain name to Punycode. Only the non-ASCII part of the domain name is converted.

File: punycode_example3.js

```
punycode = require('punycode');
console.log(punycode.toASCII('mañana.com'));
```

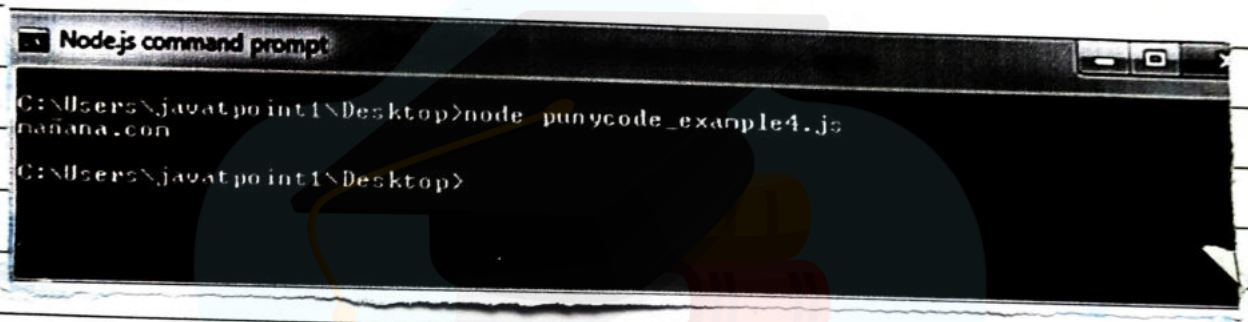


punycode.toUnicode(domain)

It is used to convert a Punycode string representing a domain name to Unicode. Only the Punycoded part of the domain name is converted.

File: punycode_example4.js

```
punycode = require('punycode');
console.log(punycode.toUnicode('xn-maana-pta.com'));
```



Node.js TTY

The Node.js TTY module contains tty-ReadStream and tty.WriteStream classes. In most cases, there is no need to use this module directly.

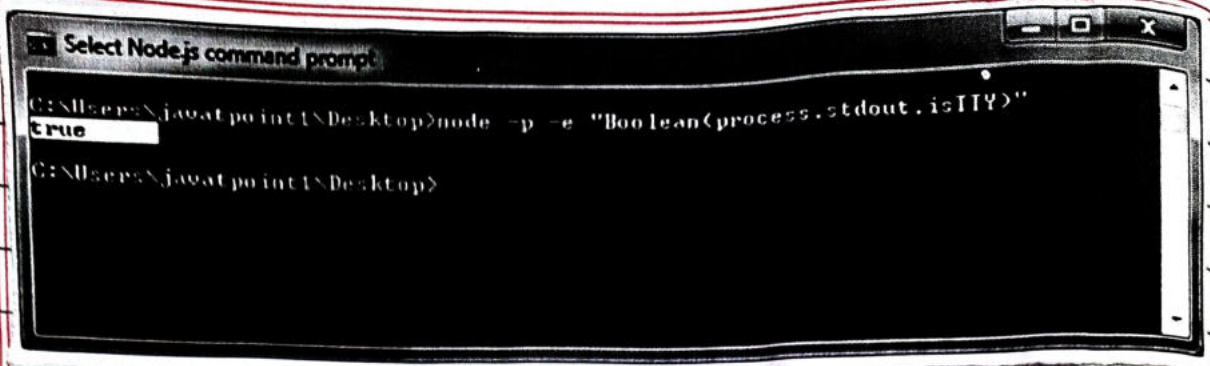
you have to use require('tty') to access this module

```
Var tty = require('tty')
```

When Node.js discovers that it is being run inside a TTY context, than:

- process.stdin will be a tty.ReadStream instance.
- process.stdout will be a tty.WriteStream instance.

To check that if Node.js is running in a TTY context, use the following command:



Class: ReadStream

It contains a net.Socket subclass that represents the readable portion of a tty. In normal circumstances, the tty.ReadStream has the only instance named process.stdin in any Node.js program (only when isatty(0) is true).

rs.isRaw:

It is a Boolean that it initialized to false. It specifies the current "raw" state of the tty.Readstream Instance.

rs.setRawMode(mode)

It is should be true or false. It is used to set the properties of the tty.ReadStream to act either as a raw device or defalut, is Raw will be set to the resulting mode.

Class: WriteStream

It contains a net.Socket subclass that represents the Writable portion of a tty. In normal circumstances, the tty.Writestream has the only instance named process.stdout it any Node.js program (only when isatty(1) is true).

Resize event: This event is used When either of the columns or raws properties has changed.

```
process.stdout.on('resize', () => {
  console.log('screen size has changed!');
  console.log('${process.stdout.columns}x${process.
                                    stdout.rows}');
});
```

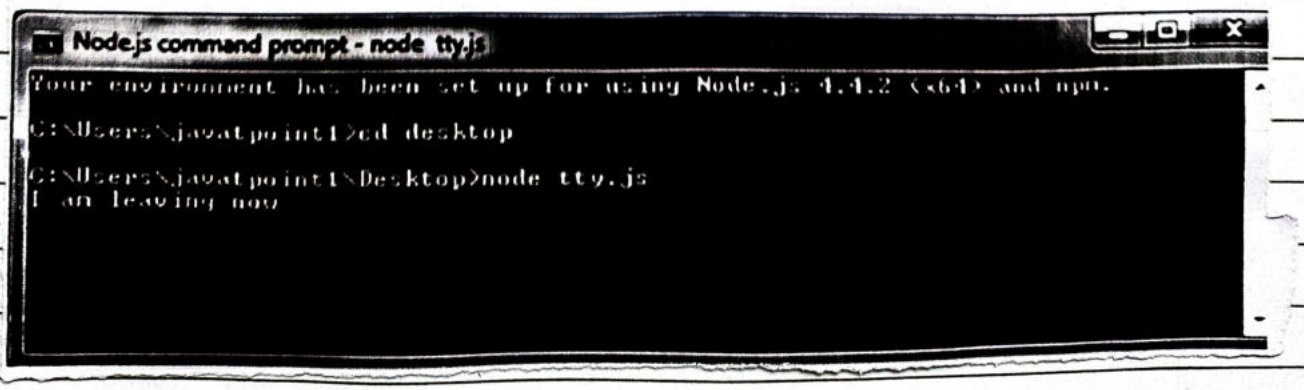
ws.columns:
It is used to give the number of columns the TTY currently has. This property gets updated on 'resize' events.

ws.rows:
It is used to give the number of rows the TTY currently has. This property gets updated on 'resize' events.

Node.js TTY Example

File: tty.js

```
Var tty = require ('tty');
process.stdin.setRawMode(true);
process.stdin.resume();
 console.log('I am leaving now');
process.stdin.on('keypress', function (char, key) {
  if (key && key.ctrl && key.name == 'c') {
    process.exit()
  }
});
```



Node.js Web Module

What is Web Server

Web Server is a software program that handles HTTP request sent by HTTP clients like Web browsers, and returns Web pages in response to the clients. Web servers usually respond with ht documents along with images, style sheets an scripts.

Web Application Architecture

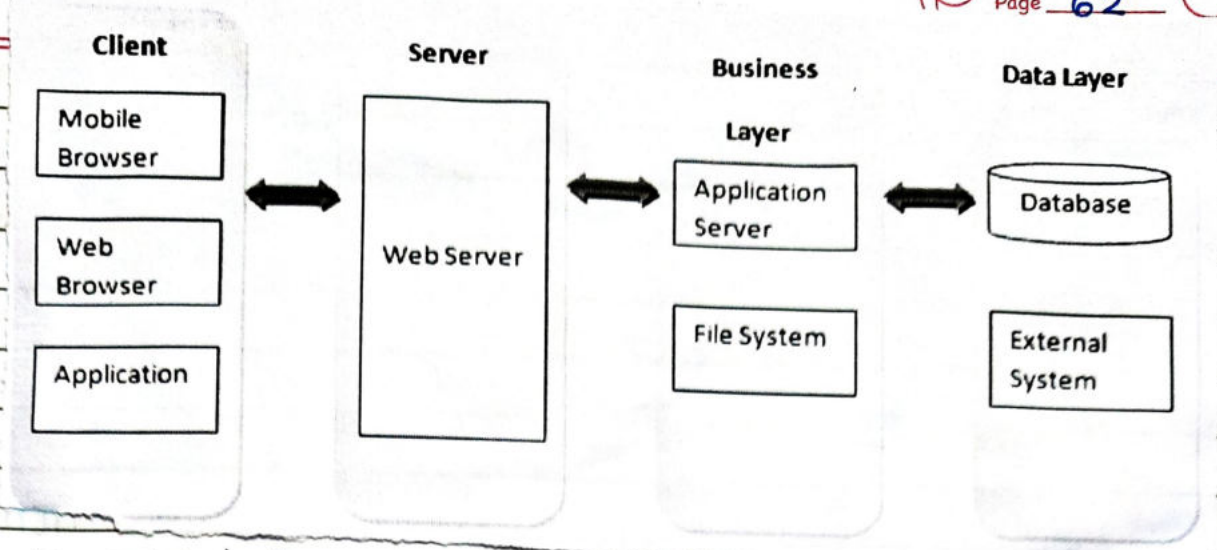
A Web application can be devided in 4 layers:

- **Client Layer:**
  The client layer contains Web browsers, mobile browsers or applications which can make HTTP request to the Web Server.

- **Server layer:**
  The Server layer contains Web server which can inter cepts the request made by clients and pass them the response.

- **Business Layer:**
  The Business layer contains application server which is utilized by Web server to do required processing. This layer interacts with data layer via data base or some external programs.

- **Data Layer:**
  The data layer contains database or any sourc of data.



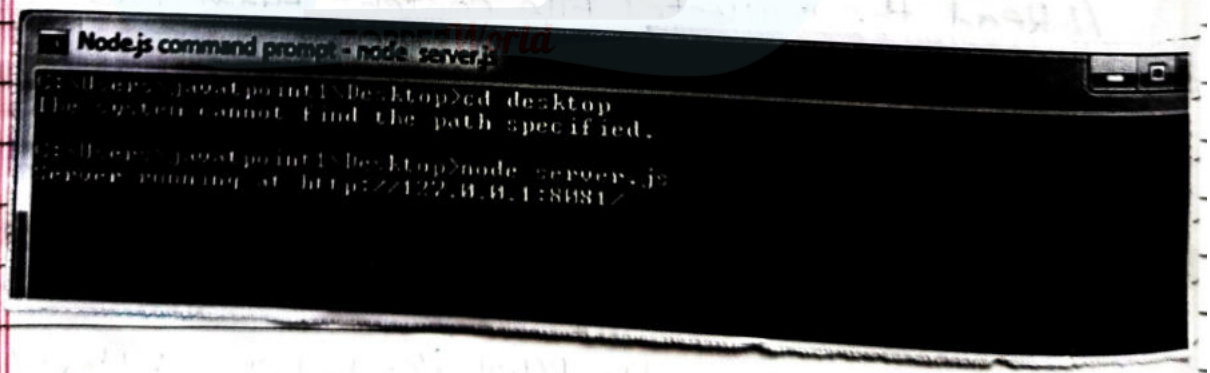
Creating Web Server using Node.js

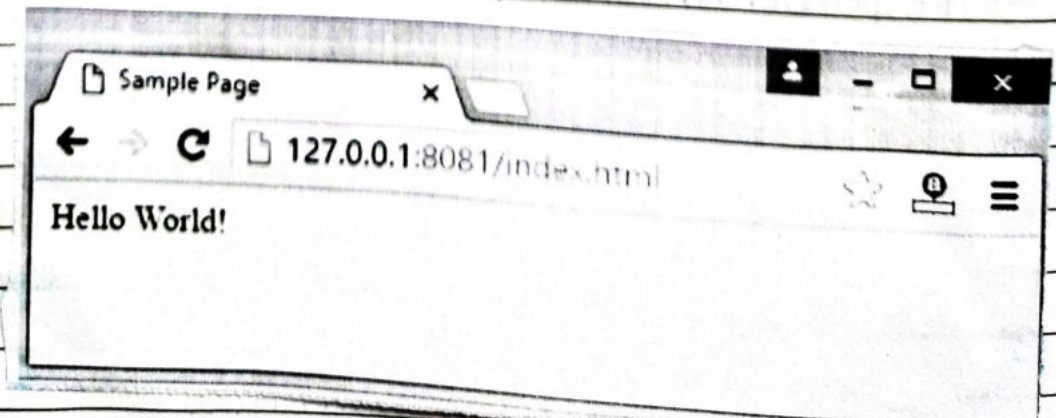
```
Var http = require ('http');
Var fs   = require ('fs');
Varurl   = require ('url');
// Create a Server
http.createServer (function(request, response){
   // Parse the request containing file name
   Var parthname = Url.parse(request.url).pathname;
   // Print the name of the file for Which request is mode.
   Console.log ("Request for" + pathname + "received.");
   // Read the requested file content from file system.
   fs.readFile(pathname.substr(1), function (err, data){
    if (err) {
      Console.log (err);
      // HTTP Status : 404: NOT FOUND
      // Content Type : text/plain
      response.WriteHead (404, {'Content-Type': 'text/html'});
    }else{
      // Page found
      // HTTP Status : 200: Ok
      // Content Type: text/plain
      response.WriteHead(200, {'Content-Type': 'text/html'});
      // Write the content of the file to response body
       response.Write(data.toString());
    }
```

```
    // Send the response body
    response.end();
  });
}).listen(8081);
// Console will print the message.
console.log("Server running at http://127.0.0.1:8081/");
```

Next, create an html file named index.html having the following in the same directory where you created server.js

```
<html>
<head>
<title> Sample page </title>
</head>
<body>
Hello World!
</body>
</html>
```





Node.Js Create Connection With MySQL

Create Connection

Create a folder named "DBexample". In that folder Create a js file named "connection.js" having the following code:

```
Var mysql = require('mysql');
Var con = mysql.createConnection({
  host: "localhost",
  User: "root",
  password: "12345"
});

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

```
root@aspire-pc: /home/user/Desktop/DBexample
root@aspire-pc:/home/user/Desktop/DBexample# node connection.js
Connected!
```

Node.js MySQL Create Database

CREATE DATABASE statement is used to create a database in MySQL.

Example

For creating a database named "javatpoint". Create a js file named Javatpoint.js having the following data in DBexample folder.

```
Var mysql = require('my sql');
Var con = mysql.createConnection({
 host: "localhost",
```

```
user: "root",
password: "12345"
});
con.connect(function(err) {
if(err) throw err;
Console.log("Connected!");
con.query("CREATE DATABASE javatpoint", function(err,
                                    result) {
if (err) throw err;
Console.log("Connected!");
con.query log("Database created");
});
});
```

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user# cd Desktop
root@aspire-pc:/home/user/Desktop# cd MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node createdatabase.js
Database created!
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

NOde.js MySQL Create Table

CREATE TABLE command is used to create a table in MySQL. You must make it sure that you define the name of the database When you create the connection.

Example

For creating a table named "employees". Create a js file named employee.js having the following data in DBexample folder.

```
Var mysql = require('mysql');
Var con = mysql.Create Connection({
host: "localhost",
```

```
user: "root",
password: "12345"
data base: "javatpoint"
});
con.connect(function(err){
if(err) throw err;
console.log("Connected!");
Var sql ="CREATE TABLE employees(id INT, name VARCHAR(255),
          age INT(3), city VARCHAR(255))";
con.query(sql, function (err, result){
 if (err) throw err;
 Console.log ("Table created");
 });
 });
```

```
root@aspire-pc: /home/user/Desktop/DBexample
root@aspire-pc:/home/user/Desktop# cd DBexample
root@aspire-pc:/home/user/Desktop/DBexample# node employees.js
Connected!
Table created
```

Create Table Having a Primary key

Create Primary key in new table:
Let's create a new table named "employee2" having id as primary key.
Create a js file named employee2.js having the following data in DBexample folder.

```
Var mysql = require('mysql');
Var con = mysql.createconnection({
host: "localhost",
user: "root",
password: "12345",
data base: "javatpoint"
});
```

```
Con.connect(function(err){
if(err) throw err;
Console.log("Connected!");
Var sql = "CREATE TABLE employee2 (id INT PRIMARY
          KEY, name VARCHAR(255), age INT(3), city VARC
Con.query(sql, function(err, result){
 if(err) throw err;
 console.log("Table created");
 });
 });
```

```
root@aspire-pc: /home/user/Desktop/DBexample
root@aspire-pc:/home/user/Desktop/DBexample# node employee2.js
Connected!
Table created
```

Add columns in existing Table:

ALTER TABLE statement is used to add a column in an existing table. Take the already created table "employee2" and use a new column Salary. Replace the data of the "employee2" table with the following data:

```
Var mysql = require('mysql');
Var con = mysql.createConnection({
 host: "localhost";
 user: "root"
  password: "12345",
  database: "javatpoint"
 });
  Con.connect(function(err){
  if(err) throw err;
   console.log("Connected!");
```

```
Var sql = "ALTER TABLE employee2 ADD COLUMN salary INT(10)";
con.query(sql, function (err, result) {
if (err) throw err;
console.log("Table alerted");
});
});
```

```
root@aspire-pc: /home/user/Desktop/DBexample
root@aspire-pc:/home/user/Desktop/DBexample# node employee2.js
Connected!
Table altered
```

Nodejs My SQL Insert Records

INSERT INTO Statement is used to insert records in MySQL.

Example

Insert Single Record:

Insert records in "employees" table.

Create a js file named "insert" in D.B example folder and put the following data into it:

```
Var mysql = require('mysql');
Var con = mysql.createConnection({
host: "localhost",
user: "root",
password: "12345"
database: "javatpoint"
});
con.connect(function(err){
if (err) throw err;
console.log("connected!");
Var sql = "INSERT INTO employees (id, name, age, city)
        VALUES ('1', 'Ajeet kumar', '27', 'Allahabad')";
```

```
con.query (sql, function (err, result){
if (err) throw err;
Console.log ("1 record inserted");
});
```

Now open command terminal and run the following command:

Node insert.js

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
oot@aspire-pc:/home/user/Desktop/MongoDatabase# node insert.js
 record inserted
oot@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Check the inserted record by using SELECT query:

SELECT * FROM employees;

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node insertall.js
Number of records inserted: 4
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```


The Result Object

When executing the insert() method a result object is returned. The result object contains information about the insertion.

It is looked like this:

```
root@aspire-pc: /home/user/Desktop/DBexample
^C
root@aspire-pc:/home/user/Desktop/DBexample# node select.js
[ RowDataPacket { id: 1, name: 'Ajeet Kumar', age: 27, city: 'Allahabad' },
  RowDataPacket { id: 2, name: 'Bharat Kumar', age: 25, city: 'Mumbai' },
  RowDataPacket { id: 3, name: 'John Cena', age: 35, city: 'Las Vegas' },
  RowDataPacket { id: 4, name: 'Ryan Cook', age: 15, city: 'CA' } ]
```

Node.js MySQL Update Records

The UPDATE command is used to update records in the table.

Example

Update city in "employees" table where id is 1.

Create a file named "update" in DBexample folder and put the following data into it.

```
Var mysql = require ('mysql');
Var con = mysql.createConnection({
host: "localhost",
user: "root"
password: "12345"
database: "javatpoint"
});
con.connect(function(err){
if (err) throw err;
Var sql = "UPDATE employees SET city = 'Delhi' WHERE
          city = 'Allahabad'";
con.query (sql, function (err, result) {
if (err) throw err;
console.log(result.affectedRows + "record (s)updated")
});
});
```

Now open command terminal and run the following command:

```
Node update.js
```

It will change the city of the id 1 is to Delhi which is prior Allahabad.

```
root@aspire-pc: /home/user/Desktop/DBexample
^C
root@aspire-pc:/home/user/Desktop/DBexample# node update.js
1 record(s) updated
```

Node.js My SQL Delete Records

The DELETE FROM command is used to delete records from the table.

Example

Delete employee from the table employees where City is Delhi.

Create a js file named "delete" in DBexample folder and put the following data into it:

```
Var my sql = require ('mysql');
 Var con = mysql.createconnection({
 host:"localhost"
 user: "root",
 password : "12345",
 database: "javatpoint"
 });
 con.connect(function(err){
 if (err) throw err;
 Var sql = "DELETE FROM employees WHERE city 'Delhi";
 con.query(sql, function (err, result){
 if(err) throw err;
 console.log("Number of records deleted:" + result.
                affectedRows);
 });
 });
```

Now open command terminal and run the following command:

```
Node delete.js
```

```
root@aspire-pc: /home/user/Desktop/DBexample
:;
root@aspire-pc:/home/user/Desktop/DBexample# node delete.js
Number of records deleted: 1
```

Node.js My SQL Select Records

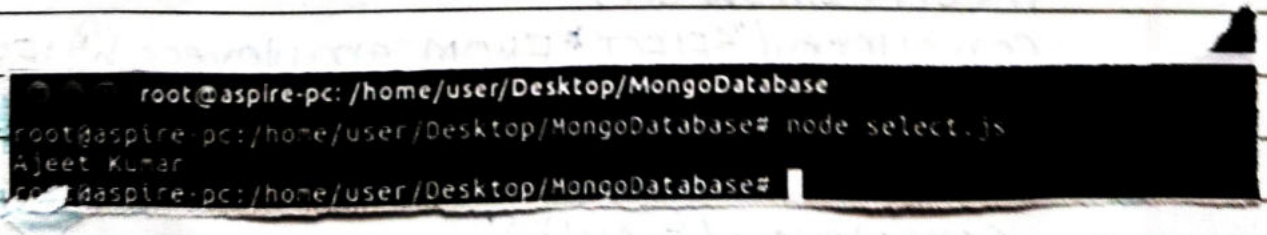
Example

Retrieve all data from the table "employees".
Create a js file named select.js having the following data in DBexample folder.

```
Var mysql = require ('mysql');
Varcon = mysql.createConnection ({
 host: "localhost",
 user: "root",
 password: "12345",
 database: "javatpoint"
 });
 con.connect(function(err){
 if(err) throw err;
 con.query("SELECT * FROM employees", function(err,result){
 if(err) throwerr:
 Console.log(result);
 });
 });
```

Now open command terminal and run the following command.

Node select.js



You can also use the statement:

SELECT * FROM employees;

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node selectall.js
[ { _id: 59104e062f60cd3366ceb7da,
    name: 'Ajeet Kumar',
    age: '28',
    address: 'Delhi' },
  { _id: 59104e062f60cd3366ceb7db,
    name: 'Mahesh Sharma',
    age: '25',
    address: 'Ghaziabad' },
  { _id: 59104e062f60cd3366ceb7dc,
    name: 'Tom Moody',
    age: '31',
    address: 'CA' },
  { _id: 59104e062f60cd3366ceb7dd,
    name: 'Zahira Wasim',
    age: '19',
    address: 'Islamabad' },
  { _id: 59104e062f60cd3366ceb7de,
    name: 'Jack Ross',
    age: '45',
    address: 'London' } ]
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Node.js MySQL SELECT Unique Record

(WHERE Clause)

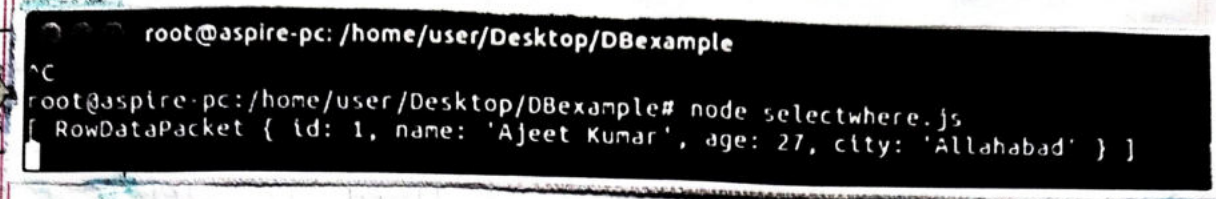
Retrieve a unique data from the table "employee"
Create a js file named selectwhere.js having the following data in DBexample folder.

```
Var mysql = require ('mysql');
Var con = mysql.createConnection({
 host: "localhost",
 user : "root",
 password: "12345"
 database: "javatpoint"
 });
 con.connect(function(err) {
 if (err) throw err;
 con.querry("SELECT * FROM employess WHERE id =
            '1', function(err, result) {
 If (err) throw err;
  console.log (result);
 });
 });
```

Now open command terminal and run the

following command:

Node selectWhere.js



Node.js My SQL Select Wildcard

Retrieve a unique data by using Wildcard from the table "employees".

Create a js file named selectwildcard.js having the following data in DBexample folder.

```
Var mysql = require('mysql');
Varcon = mysql.createConnection({
host: "localhost",
user: "root"
password: "12345"
database: "javatpoint"
});
con.connect(function(err) {
if (err) throw err.
con.query("SELECT * FROM employees WHERE city LIKE
          'A%'", function (err, result) {
if (err) throw err;
console.log (result);
});
});
```

Now open command terminal and run the following command

Node selectWildcard.js

It will retrieve the record where City start with A.

```
root@aspire-pc: /home/user/Desktop/DBexample
^C
root@aspire-pc:/home/user/Desktop/DBexample# node selectwildcard.js
[ RowDataPacket { id: 1, name: 'Ajeet Kumar', age: 27, city: 'Allahabad' } ]
```

Node.js MySQL Drop Table

The DROP TABLE command is used to delete or drop a table.

Lets drop a table named employee2.

Create a js file named "delete" in DBexample folder and put the following data into it:

```
Var mysql = require('mysql');
Var con = mysql.createConnection({
host: "localhost",
user: "root"
password: "12345",
database: "javatpoint"
});
Con.connect(function(err){
if(err) throw err;
Var sql = "DROP TABLE employee2";
Con.query(sql, function(err, result){
if(err) throw err;
console.log("Table deleted");
});
});
```

Now open command terminal and run the following command:

```
Node drop.js
```

```
root@aspire-pc: /home/user/Desktop/DBexample
^C
root@aspire-pc:/home/user/Desktop/DBexample# node drop.js
Table deleted
```

Node.js Create Connection With Mongo DB

MongoDb is a NOSQL database. It can be used with Node.js as a database to insert and retrieve data. Use the following command to start MongoDB Services:

```
Service mongodb start
```

Now, connection is created for further operations.

Node.js Mongo DB Create Database

To create a database in MongoDB. First create a Mongo Client and specify a connection URL with the correct ip address and the name of the database which you want to create.

Example

Create a folder named "MongoDatabase" as a database. Suppose you create it on Desktop. Create a file named "Createdatabase.js" within that folder and having the following code:

```
Var MongoClient = require('mongodb').MongoClient;
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function(err, db) {
if (err) throw err;
console.log("Database created!");
db.close();
});
```

Now open the command terminal and set the path where MongoDatabase exists. Now execute the following command:

Node create data base.js

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user# cd Desktop
root@aspire-pc:/home/user/Desktop# cd MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node createdatabase.js
Database created!
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Now database is created.

Node.js MongoDB Create Collection

MongoDB is a NoSQL database so data is stored in collection instead of table. Create Collection method is used to create a collection in MongoDB.

Example

Create a collection named "employees".

Create a js file named "employees.js", having the following data:

```
Var Mongo client = require('mongodb').MongoClient;
Var url = "mongodb://localhost:27017/Mongo Database";
Mongo Client.connect (url, function(err, db) {
if(err) throw err;
db.create Collection("employees", function(err, res) {
if(err) throwerr;
console.log("Collection is created!");
db.close();
});
});
```

Open the Command terminal and run the following command:

Node employees.js

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node employees.js
Collection is created!
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Now the collection is created.

Node.js MongoDB Insert Record

The insertOne method is used to insert record in MongoDB's collection. The first argument of the insertOne method is an object which contains the name and value of each field in the record you want to insert.

Example

(Insert Single record)

Insert a record in "employees" collection.

Create a js file named "insert.js", having the following code:

```
Var MongoClient = require('mongodb').MongoClient;
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function(err,db){
if (err) throw err;
Var myobj={name:"Ajeet kumar", age:"28", address: "Delhi"};
db.collection("employees").insertone(myobj, function(err,res){
If(err) throw err;
console.log("1 record inserted");
db.close();
});
});
```

Open the command terminal and run the following command:

```
Node insert.js
```

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node insert.js
1 record inserted
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Now a record is inserted in the Collection.

Insert Multiple Records

You can insert multiple records in a collection by using insert()method. The insert()method uses array of objects which contain the data you want to insert.

Example

Insert multiple records in the collection named "employees".

Create a js file name insertall.js having the following Code:

```
Var MongoClient = require ('mongodb').MongoClient;
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function(err, db) {
if (err) throw err;
Var myobj = [
{name:"Mahesh sharma", age:"25", address:"Ghaziabad"},
{name:"Tom Moody", age:"31", address: "CA"},
{name: "Zahira Wasim", age:"19", address:"Islamabad"},
];
db.collection("customers").insert(myobj, function(err, res) {
if (err) throw err;
console.log("Number of records inserted: " + res.insertedCount);
db.close();
});
});
```

Open the command terminal and run the following command:

Node. insertall.js

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node insertall.js
Number of records inserted: 4
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

You can see here 4 records are inserted.

Node.js MongoDB Select Record

The findOne() method is used to select a single data from a collection in MongoDB. This method returns the first record of the collection.

Example

(Select Single Record)

Select the first record from the "employees" collection.

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

```
Var http = require('http');
Var MongoClient = require('mongodb').MongoClient;
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function(err, db){
  if(err) throw err;
  db.Collection("employees").fineOne({}, function(err, result){
  if(err) throw err;
  Console.log(result.name);
  db.close();
  });
});
```

open the command terminal and run the following command:

Node Select.js

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node remove.js
1 record(s) deleted
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Select Multiple Records

The find() method is used to select all the records from collection in mongoDB.

Example

Select all records from "employees" collection.

Create a js file named "selectall.js", having the following code:

```
Var Mongo client = require ('mongo DB'). Mongoclient;
Var url = "mongo db://localhost:27017/Mongo Database";
Mongoclient.connect (url, function (err, db) {
  if (err) throw err;
  db.collection ("employees").find({}).to array(function
                                    (err, result) {
    if(err) throw err;
    console.log (result);
    db.close();
  });
});
```

Open the command terminal and run the following command:

```
Node selectall.js
```

Node.js MongoDB Filter Query

The find() method is also used to filter the result on a specific parameter. You can filter the result by using a query object.

Example

Filter the records to retrieve the specific employee whose address is "Delhi".

Create a js file named "query1.js" having the following code:

```
Var http = require('http');
VarMongoClient = require('mongodb').MongoClient;
Varurl = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url,function(err,db){
if(err) throw err;
Varquery = {address:"Delhi"};
db.collection("employees").find(query).toArray(function
                                (err,result){
if(err) throwerr;
console.log(result);
db.close();
});
});
```

Open the Command terminal and run the following Command

```
Node query1.js
```

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node query1.js
[ { _id: 591040c52a89e8301bde229a,
    name: 'Ajeet Kumar',
    age: '28',
    address: 'Delhi' } ]
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Node.js Mongo DB Sorting

In MongoDB, the sort() method is used for sorting the results in ascending or descending order. The sort() method uses a parameter to define the object sorting order.

Value used for sorting in ascending order:

```
[name:1]
```

Value used for Sorting in descending order:

```
[name:-1]
```

Sort in Ascending Order

Example

Sort the records in ascending order by the name
Create a js file named "Sortasc.js" having the following code:

```
Var http = require('http');
Var Mongo client = require('mongodb').mongo client
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function(err, db) {
if(err) throw err;
Var mysort = {name:1};
db.collection("employees").find().sort(mysort).toArray
                    (function(err, result) {
if(err) throw err;
Console.log(result);
db.close();
});
});
```

Open the Command terminal and run the following comma

```
Node sortasc.js
```

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node sortasc.js
[ { _id: 591040c52a89e8301bde229a,
    name: 'Ajeet Kumar',
    age: '28',
    address: 'Delhi' },
  { _id: 59104e062f60cd3366ceb7dd,
    name: 'Juck Ross',
    age: '45',
    address: 'London' },
  { _id: 59104e062f60cd3366ceb7da,
    name: 'Mahesh Sharma',
    age: '25',
    address: 'Ghaziabad' },
  { _id: 59104e062f60cd3366ceb7db,
    name: 'Tom Moody',
    age: '31',
    address: 'CA' },
  { _id: 59104e062f60cd3366ceb7dc,
    name: 'Zahira Wasim',
    age: '19',
    address: 'Islamabad' } ]
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```

Node.js Mongo DB Remove

In Mongo DB, you can delete records or documents by using the remove() method. The first parameter of the remove() method is a query object which specifies the document to delete.

Example

Remove the record of employee whose address is Ghaziabad.

Create a js file named "remove.js", having the following code:

```
Var http = require('http');
Var MongoClient=require('mongodb').Mongo client;
Var url = "mongodb://localhost:27017/MongoDatabase";
MongoClient.connect(url, function (err, db){
if (err) throw err;
Var myquery = {address: 'Ghaziabad'};
db.Collection("employess").remove(myquery, function(err, obj){
if (err) throw err;
Console.log(obj.result.n +"record(s) deleted");
db.close();
});
});
```

Open the command terminal and run the following command:

```
Node remove.js
```

```
root@aspire-pc: /home/user/Desktop/MongoDatabase
root@aspire-pc:/home/user/Desktop/MongoDatabase# node insertall.js
Number of records inserted: 4
root@aspire-pc:/home/user/Desktop/MongoDatabase#
```