NODE JS INTERVIEW QUESTIONS FOR BEGINNERS TO EXPERIENCED

1) What is Node.js?

- Node.js is a web application framework built on Google Chrome's JavaScript Engine(V8 Engine). Node.js comes with runtime environment on which a Javascript based script can be interpreted and executed (It is analogous to JVM to JAVA byte code). This runtime allows to execute a JavaScript code on any machine outside a browser. Because of this runtime of Node.js, JavaScript is now can be executed on server as well. Node.js also provides a rich library of various javascript modules which eases the development of web application using Node.js to great extents.

- Node.js = Runtime Environment + JavaScript Library.

2) Explain Node.js web application architecture?

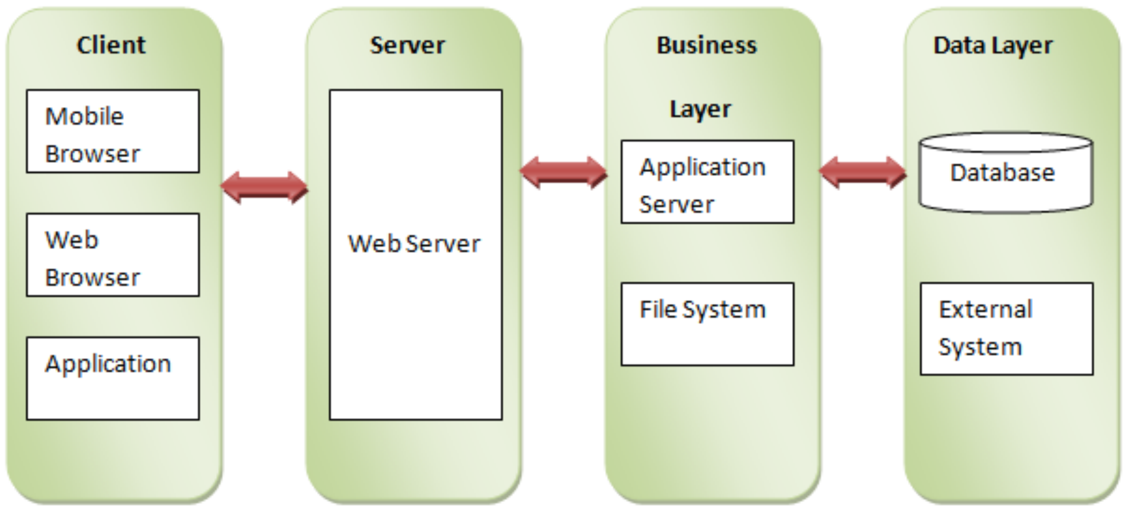
A web application distinguishes into 4 layers:

- **Client Layer :** The Client layer contains web browsers, mobile browsers or applications which can make an HTTP request to the web server.

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

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

- **Data Layer :** The Data layer contains databases or any source of data.



3) What are the benefits of using Node.js?

Following are main benefits of using Node.js

- **Asynchronous and Event Driven :** All APIs of Node.js library are asynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

- **Very Fast :** Being built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.

- **Single Threaded but highly Scalable :** Node.js uses a single threaded model with event looping. Event mechanism helps server to respond in a non-bloking ways and makes server highly scalable as opposed to traditional servers which create limited threads to handle requests. Node.js uses a single threaded program and same program can services much larger number of requests than traditional server like Apache HTTP Server.

- **No Buffering :** Node.js applications never buffer any data. These applications simply output the data in chunks.

4) Is it free to use Node.js?

Yes! Node.js is released under the MIT license and is free to use.

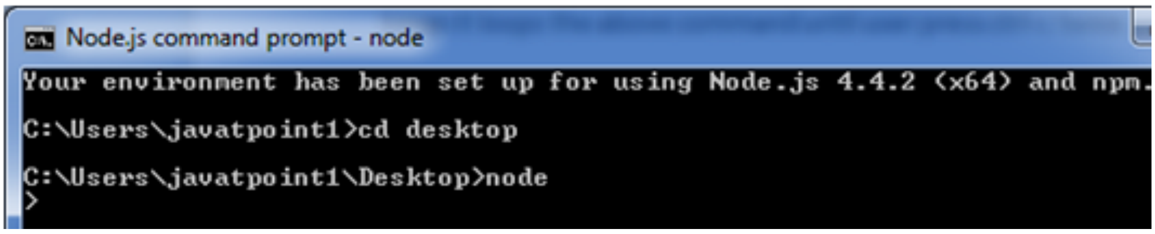
5) Is Node a single threaded application?

Yes! Node uses a single threaded model with event looping.

6) What is REPL in context of Node?

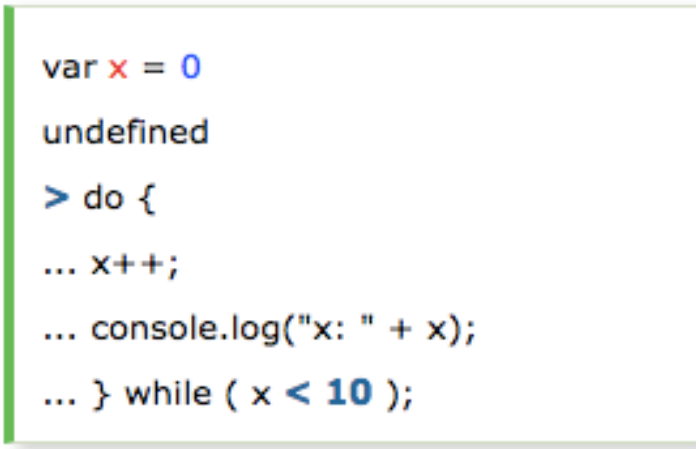
REPL stands for Read Eval Print Loop and it represents a computer environment like a window console or unix/linux shell where a command is entered and system responds with an output. Node.js or Node comes bundled with a REPL environment. It performs the following desired tasks.

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

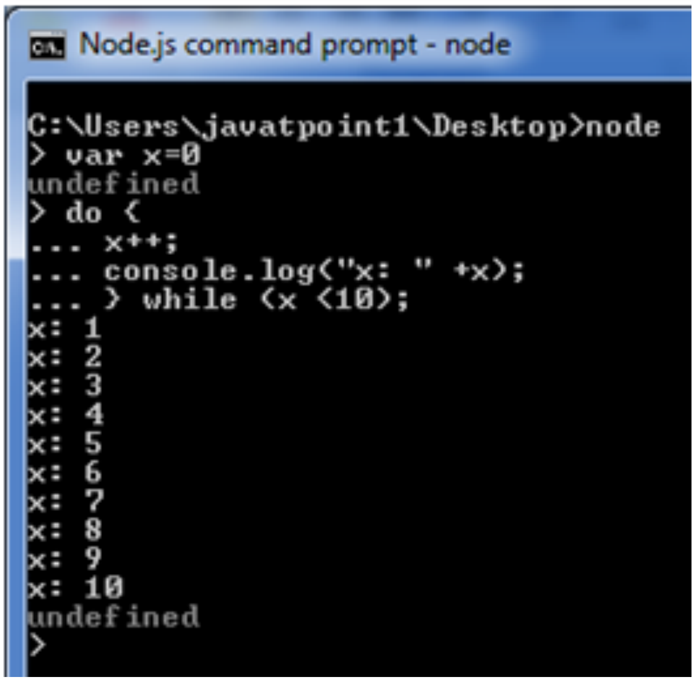


```
Your environment has been set up for using Node.js 4.4.2 (x64) and npm.
C:\Users\javatpoint1>cd desktop
C:\Users\javatpoint1\Desktop>node
>
```

- **Read** – Reads user's input, parse 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 user press ctrl-c twice.



```
var x = 0
undefined
> do {
... x++;
... console.log("x: " + x);
... } while ( x < 10 );
```



Node.js command prompt - node

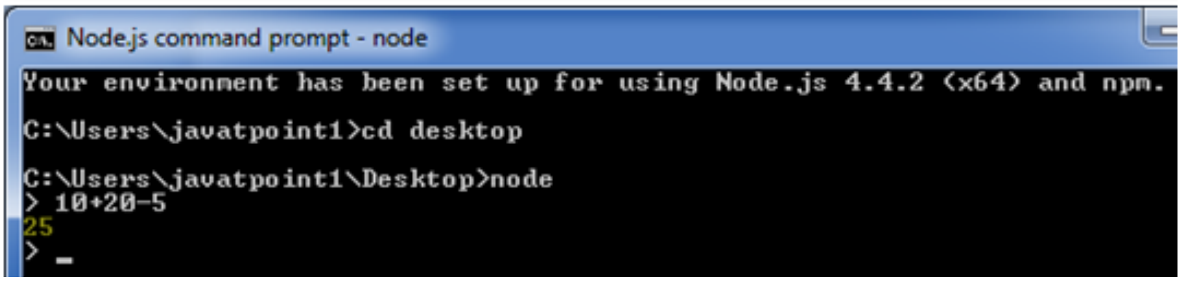
```
C:\Users\javatpoint1\Desktop>node
> var x=0
undefined
> do {
... x++;
... console.log("x: " +x);
... } while (x <10);
x: 1
x: 2
x: 3
x: 4
x: 5
x: 6
x: 7
x: 8
x: 9
x: 10
undefined
>
```

7) Can we evaluate simple expression using Node REPL?

Yes. In REPL node command prompt put any mathematical expression:

Example: >10+20-5

25

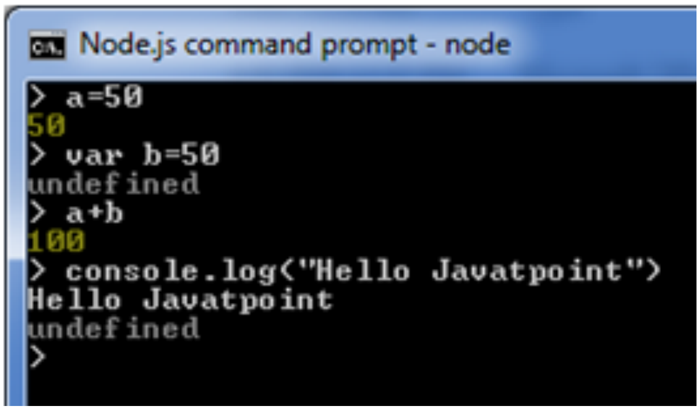


Node.js command prompt - node

```
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
> _
```

8) What is the difference of using var and not using var in REPL while dealing with variables?

Use variables to store values and print later. if var keyword is not used then value is stored in the variable and printed. Whereas if var keyword is used then value is stored but not printed. You can use both variables later.

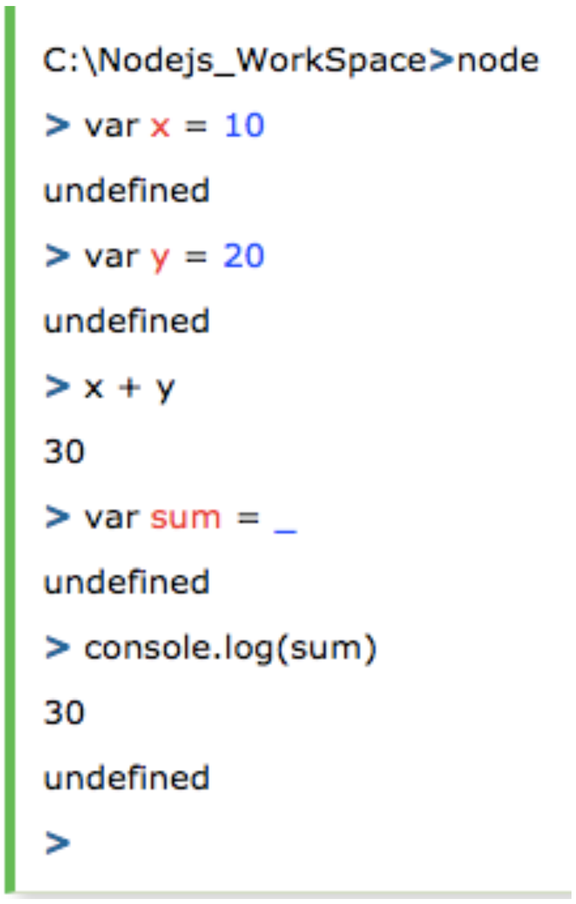


Node.js command prompt - node

```
> a=50
50
> var b=50
undefined
> a+b
100
> console.log("Hello Javatpoint")
Hello Javatpoint
undefined
>
```

9) What is the use of Underscore variable in REPL?

Use _ to get the last result.



```
C:\Nodejs_WorkSpace>node
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

10) How many types of API functions are available in Node.js?

There are two types of API functions in Node.js:

- Asynchronous, Non-blocking functions.
- Synchronous, Blocking functions.

11) What do you mean by Asynchronous API?

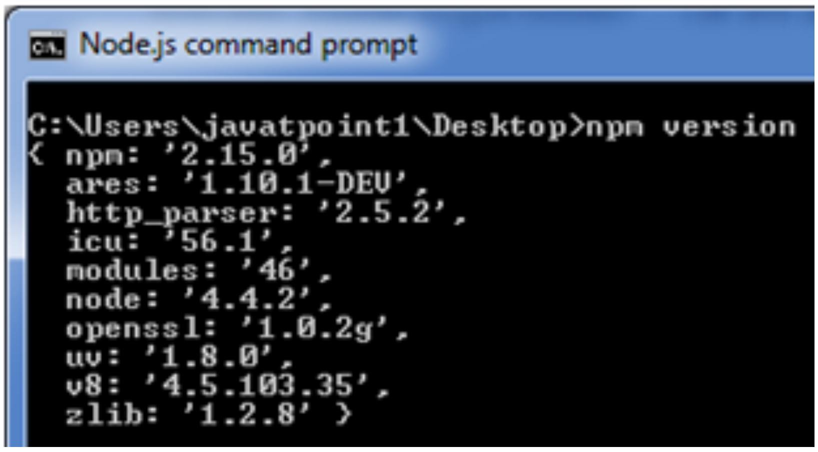
All APIs of Node.js library are aynchronous that is non-blocking. It essentially means a Node.js based server never waits for a API to return data. Server moves to next API after calling it and a notification mechanism of Events of Node.js helps server to get response from the previous API call.

12) What is npm?

npm stands for Node Package Manager. npm provides following two main functionalities:

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

```
npm  version
```



```
C:\Users\javatpoint1\Desktop>npm version
{ npm: '2.15.0',
  ares: '1.10.1-DEV',
  http_parser: '2.5.2',
  icu: '56.1',
  modules: '46',
  node: '4.4.2',
  openssl: '1.0.2g',
  uv: '1.8.0',
  v8: '4.5.103.35',
  zlib: '1.2.8' }
```

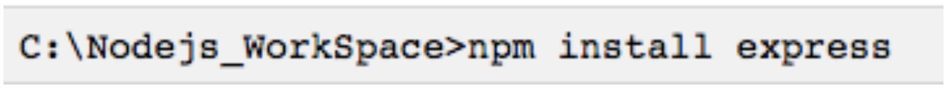
13) What is global installation of dependencies?

Globally installed packages/dependencies are stored in **<user-directory>**/npm directory. Such dependencies can be used in CLI CommandLineInterface function of any node.js but can not be imported using require in Node application directly. To install a Node project globally use -g flag.

```
C:\Nodejs_WorkSpace>npm install express -g
```

14) What is local installation of dependencies?

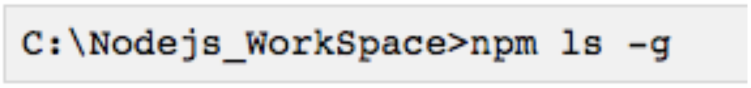
By default, npm installs any dependency in the local mode. Here local mode refers to the package installation in node_modules directory lying in the folder where Node application is present. Locally deployed packages are accessible via require. To install a Node project locally following is the syntax.



```
C:\Nodejs_WorkSpace>npm install express
```

15) How to check the already installed dependencies which are globally installed using npm?

Use the following command:



```
C:\Nodejs_WorkSpace>npm ls -g
```

16) What is Package.json?

package.json is present in the root directory of any Node application/module and is used to define the properties of a package.

17) Name some of the attributes of package.json?

Following are the attributes of Package.json

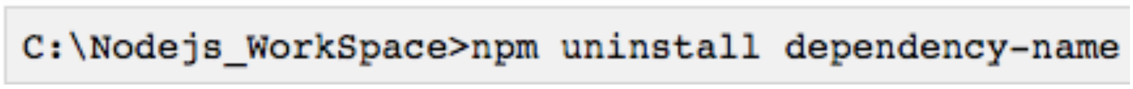
- **name** – name of the package.
- **version** – version of the package.
- **description** – description of the package.
- **homepage** – homepage of the package.
- **author** – author of the package.
- **contributors** – name of the contributors to the package.
- **dependencies** – list of dependencies. 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.

18) How to uninstall a dependency using npm?

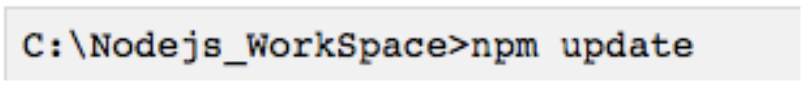
Use following command to uninstall a module:



```
C:\Nodejs_WorkSpace>npm uninstall dependency-name
```

19) How to update a dependency using npm?

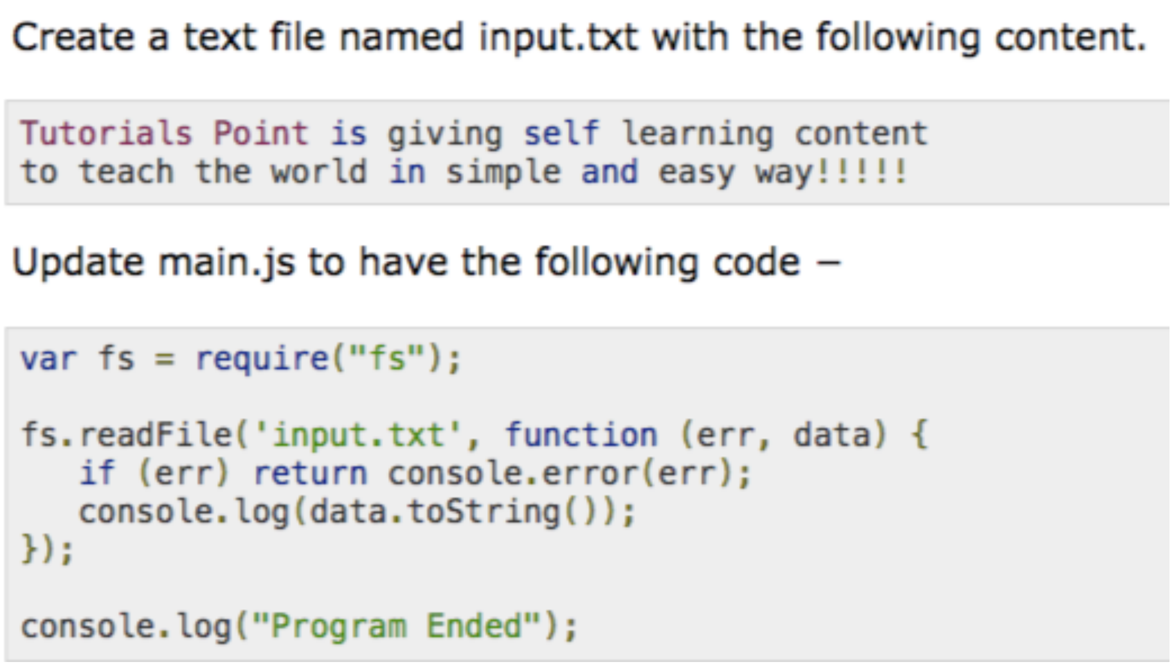
Update package.json and change the version of the dependency which to be updated and run the following command.



```
C:\Nodejs_WorkSpace>npm update
```

20) What is Callback?

Callback is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All APIs of Node are written is such a way that they supports callbacks. For example, a function to read a file may start reading file and return the control to execution environment immediately so that next instruction can be executed. Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as parameter. 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.

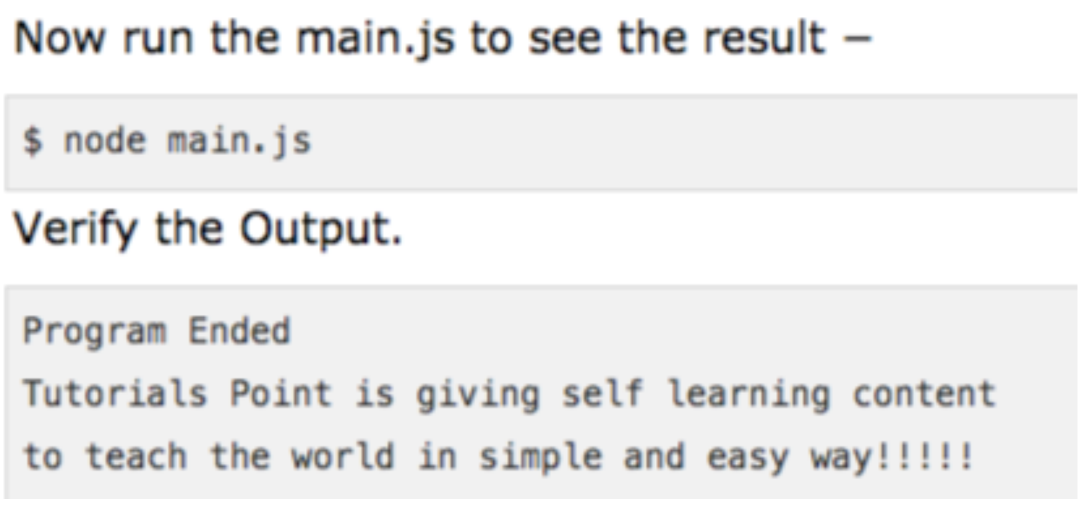


Create a text file named input.txt with the following content.

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Update main.js to have 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");
```



Now run the main.js to see the result –

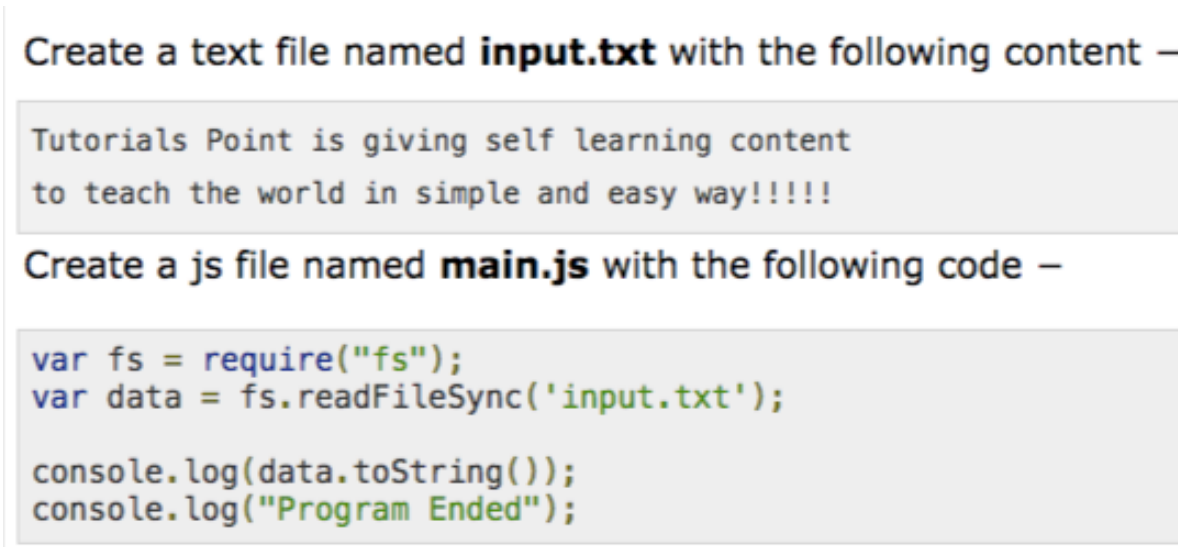
```
$ node main.js
```

Verify the Output.

```
Program Ended
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

21) What is a blocking code?

If application has to wait for some I/O operation in order to complete its execution any further then the code responsible for waiting is known as blocking code.

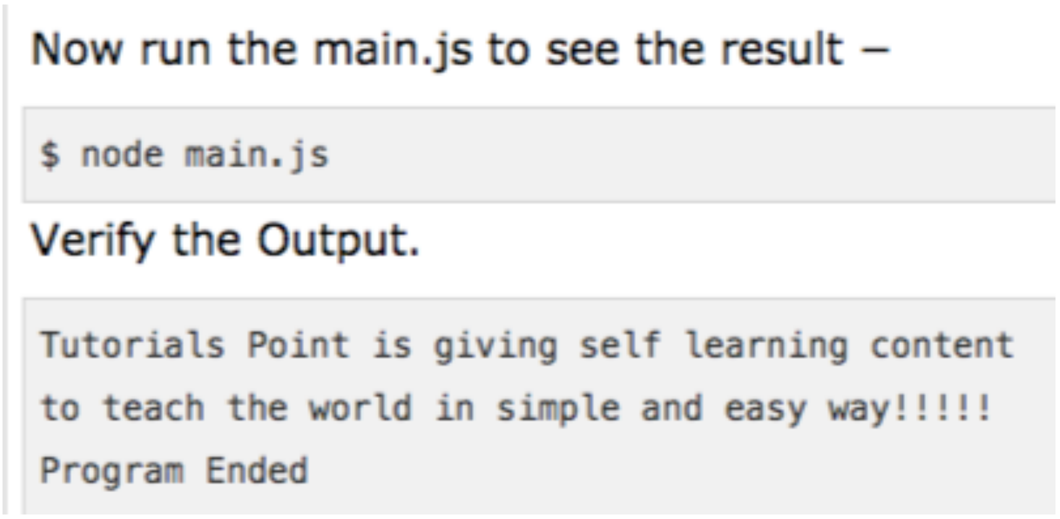


Create a text file named **input.txt** with the following content −

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Create a js file named **main.js** with the following code −

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

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



Now run the main.js to see the result −

```
$ node main.js
```

Verify the Output.

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
Program Ended
```

22) How Node prevents blocking code?

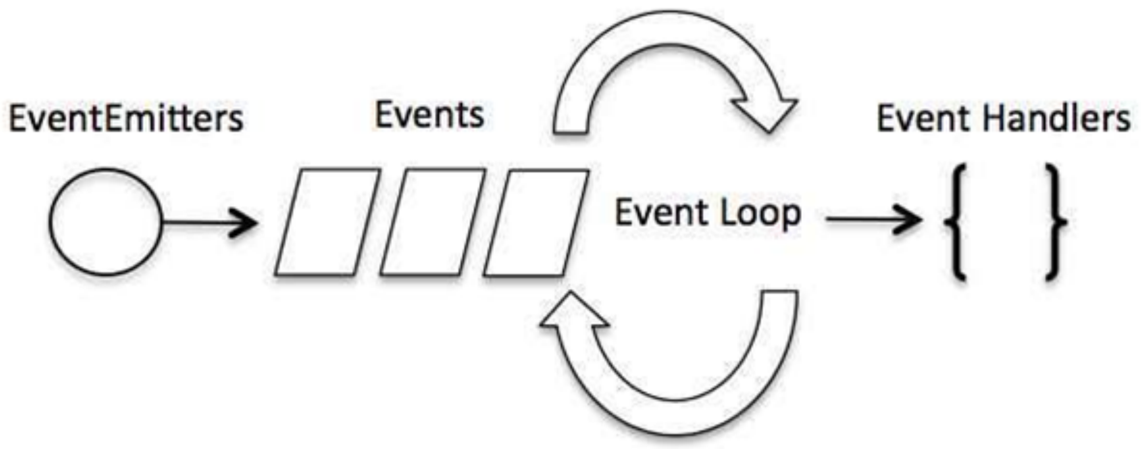
By providing callback function. Callback function gets called whenever corresponding event triggered.

23) What is Event Loop?

Node js is a single threaded application but it support concurrency via concept of event and callbacks. As every API of Node js are asynchronous and being a single thread, it uses async function calls to maintain the concurrency. Node uses observer pattern. Node thread keeps an event loop and whenever any task get completed, it fires the corresponding event which signals the event listener function to get executed.

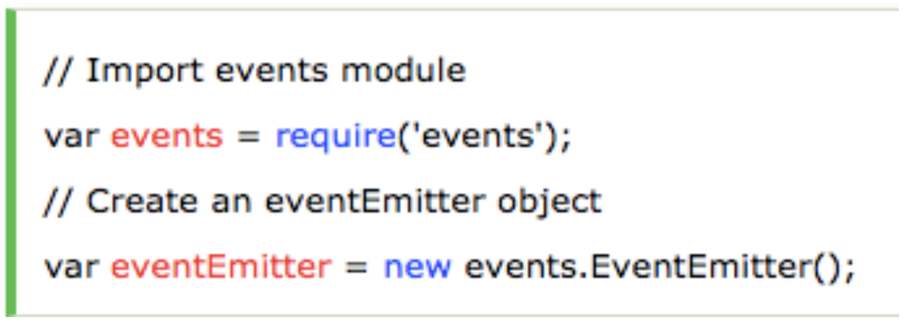
24) What is event-driven programming in Node.js?

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



25) What is Event Emitter?

EventEmitter class lies in **events** module. It is accessibly via following syntax:



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

- When an EventEmitter instance faces any error, it emits an 'error' event. When new listener is added, 'newListener' event is fired and when a listener is removed, 'removeListener' event is fired.

- EventEmitter provides multiple properties like **on** and **emit**. **on** property is used to bind a function with the event and **emit** is used to fire an event.

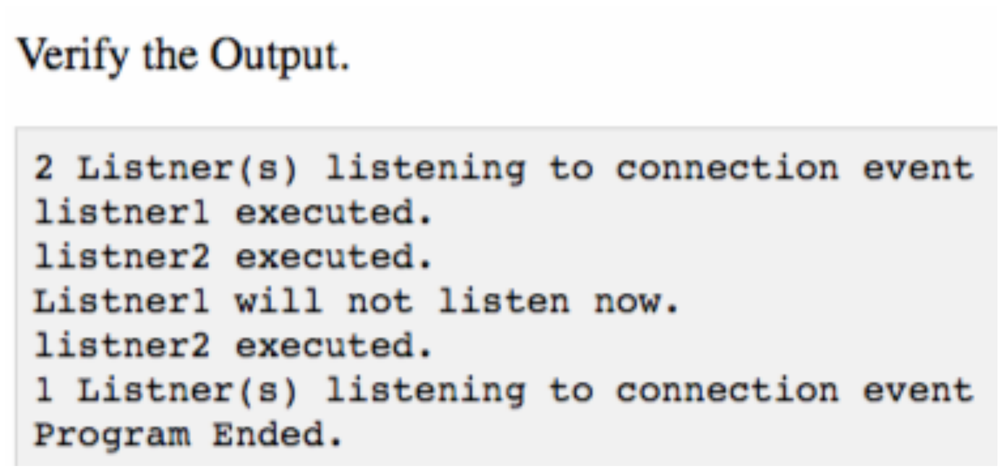
```
var events = require('events');
var eventEmitter = new events.EventEmitter();

// listener #1
var listner1 = function listner1() {
   console.log('listner1 executed.');
}

// listener #2
var listner2 = function listner2() {
   console.log('listner2 executed.');
}

// Bind the connection event with the listner1 function
eventEmitter.addListener('connection', listner1);
```

```
// Bind the connection event with the listner2 function
eventEmitter.on('connection', listner2);

var eventListeners = require('events').EventEmitter.listenerCount
   (eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

// Fire the connection event
eventEmitter.emit('connection');

// Remove the binding of listner1 function
eventEmitter.removeListener('connection', listner1);
console.log("Listner1 will not listen now.");
```

```
// Fire the connection event
eventEmitter.emit('connection');

eventListeners =
require('events').EventEmitter.listenerCount(eventEmitter,'connection');
console.log(eventListeners + " Listner(s) listening to connection event");

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

Now run the main.js to see the result –

```
$ node main.js
```



Verify the Output.

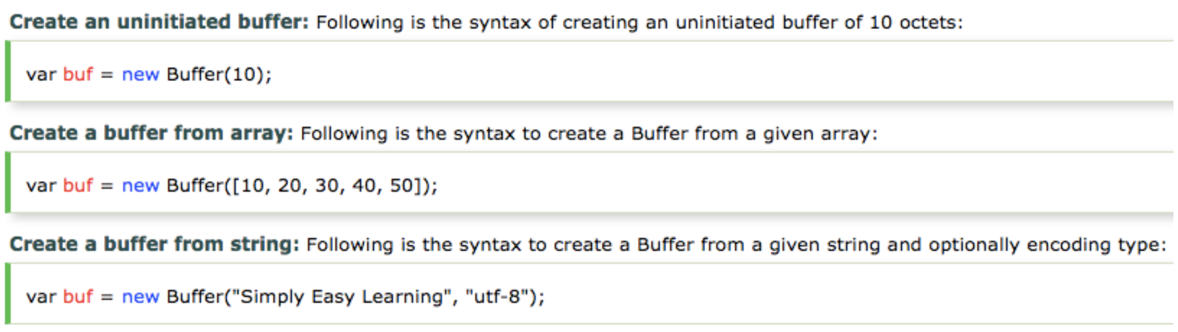
```
2 Listner(s) listening to connection event
listner1 executed.
listner2 executed.
Listner1 will not listen now.
listner2 executed.
1 Listner(s) listening to connection event
Program Ended.
```

26) What is the difference between events and callbacks in Node.js?

Although, Events and Callbacks look similar the differences lies in the fact that callback functions are called when an asynchronous function returns its result whereas event handling works on the observer pattern. Whenever an event gets fired, its listener function starts executing. Node.js has multiple in-built events available through the events module and EventEmitter class which is used to bind events and event listeners.

27) What is purpose of Buffer class in Node?

Buffer class is a global class and can be accessed in application without importing buffer module. A Buffer is a kind of an array of integers and corresponds to a raw memory allocation outside the V8 heap. A Buffer cannot be resized. There are many ways to construct a Node buffer. Following are the three mostly used methods:



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

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

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]);
```

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");
```

1. Writing to buffers

- **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.

- **Example:**

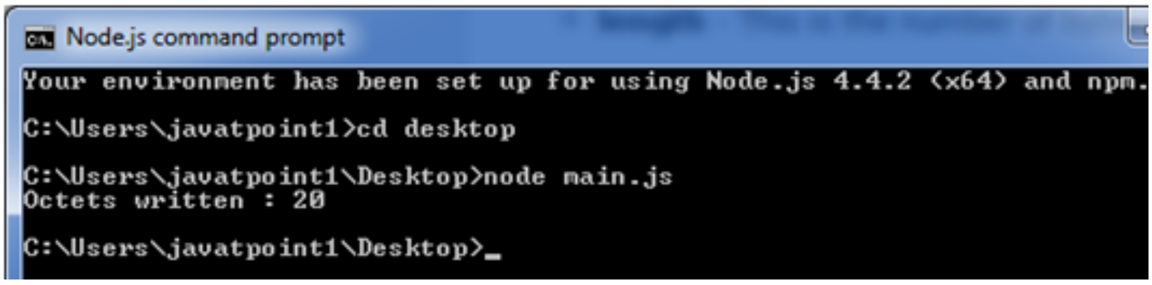
File: main.js

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

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

```
node main.js
```

Output:



```
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 main.js
Octets written : 20

C:\Users\javatpoint1\Desktop>_
```

2. Reading from buffers

- **Syntax:**

```
buf.toString([encoding][, start][, end])
```

- **Parameter explanation:**

encoding: It specifies encoding to use. 'utf8' is the default encoding

start: It specifies beginning index to start reading, defaults to 0.

end: It specifies end index to end reading, defaults is complete buffer.

Return values reading from buffers:

This method decodes and returns a string from buffer data encoded using the specified character set encoding.

- **Example:**

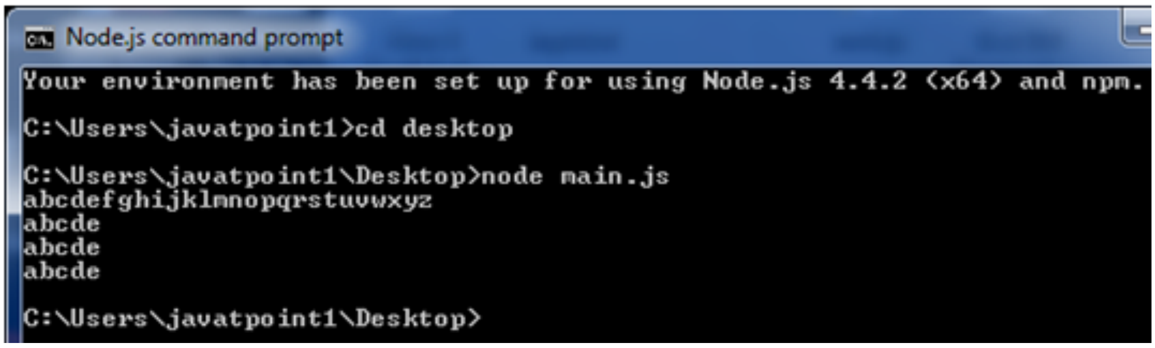
File: main.js

```
buf = new Buffer(26);
for (var i = 0 ; i < 26 ; i++) {
  buf[i] = i + 97;
}
console.log( buf.toString('ascii'));       // outputs: abcdefghijklmnopqrstuvwxyz
console.log( buf.toString('ascii',0,5));   // outputs: abcde
console.log( buf.toString('utf8',0,5));    // outputs: abcde
console.log( buf.toString(undefined,0,5)); // encoding defaults to 'utf8', outputs abcde
```

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

```
node main.js
```

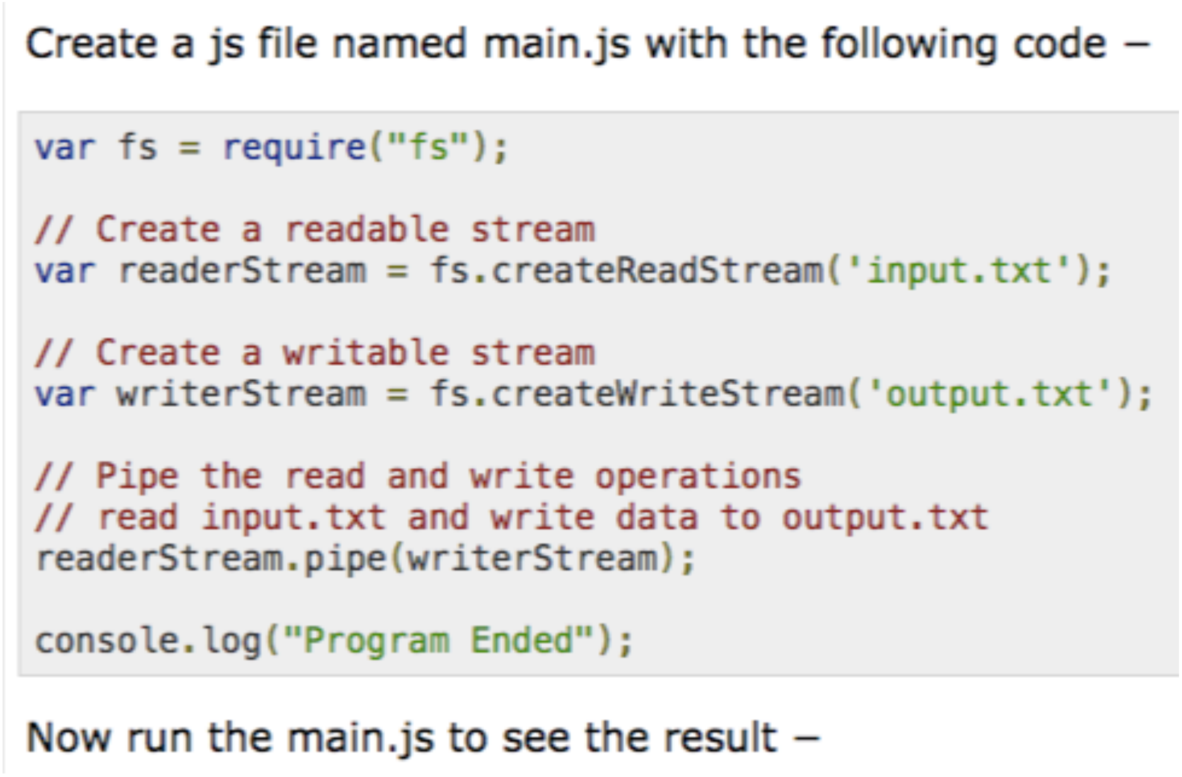
Output:



```
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 main.js
abcdefghijklmnopqrstuvwxyz
abcde
abcde
abcde

C:\Users\javatpoint1\Desktop>
```

28) What is Piping in Node?

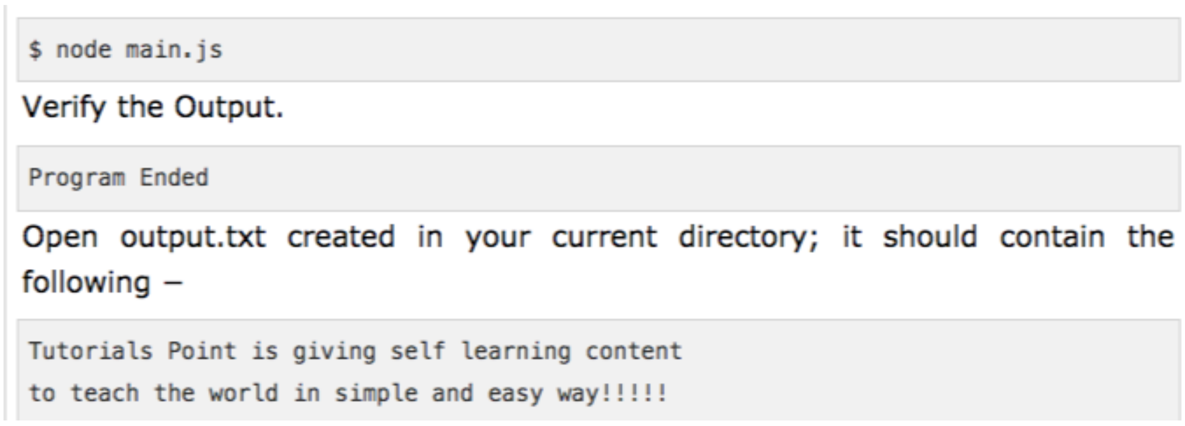
Piping is a mechanism to connect output of one stream to another stream. It is normally used to get data from one stream and to pass output of that stream to another stream. There is no limit on piping operations. Example:



Create a js file named main.js with the following code −

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

// Create a readable stream
var readerStream = fs.createReadStream('input.txt');

// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');

// Pipe the read and write operations
// read input.txt and write data to output.txt
readerStream.pipe(writerStream);

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

Now run the main.js to see the result −



```
$ node main.js
```

Verify the Output.

```
Program Ended
```

Open output.txt created in your current directory; it should contain the following –

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

29) Which module is used for file based operations?

fs module is used for file based operations.

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

30) Which module is used for buffer based operations?

buffer module is used for buffer based operations.

```
var buffer = require("buffer")
```

31) Which module is used for web based operations?

http module is used for web based operations.

```
var http = require("http")
```

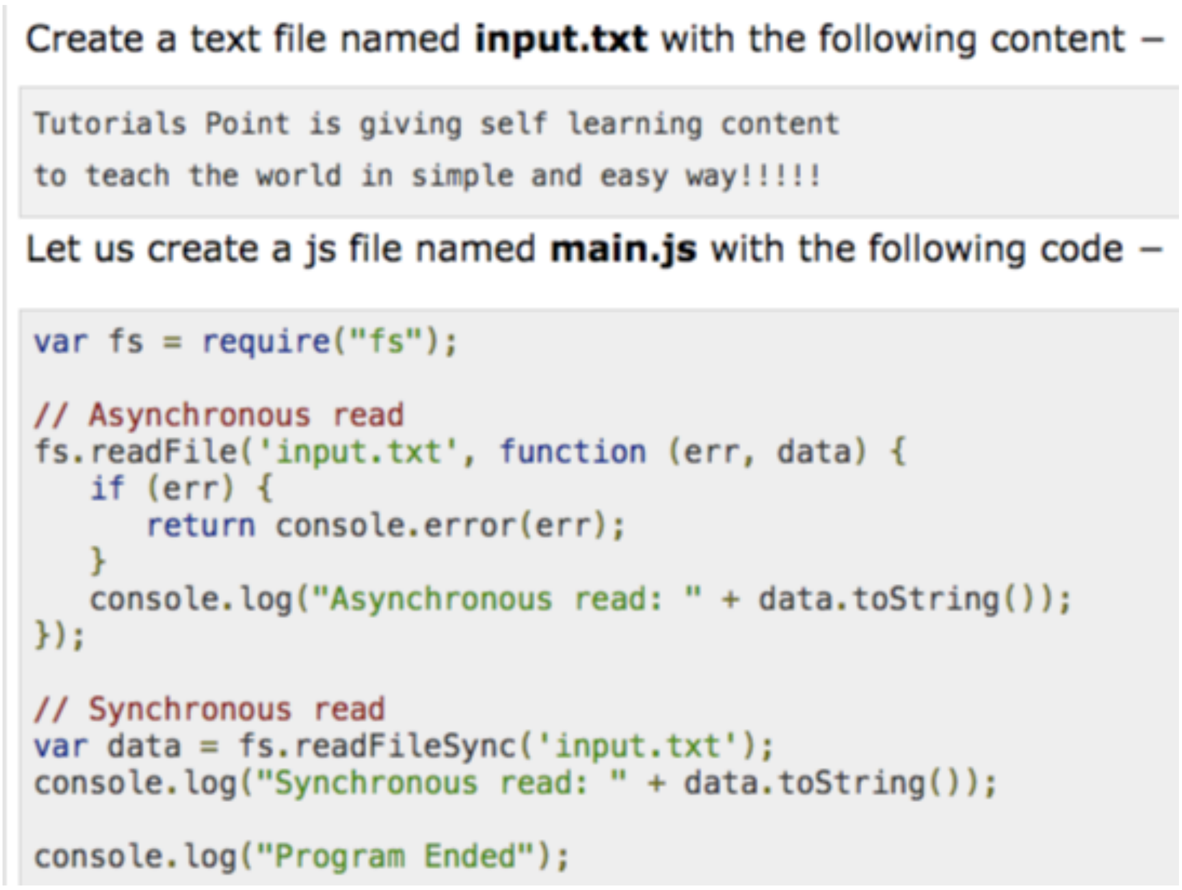
32) fs module provides both synchronous as well as asynchronous methods?

true.

33) What is difference between synchronous and asynchronous method of fs module?

Every method in fs module have synchronous as well as asynchronous form. Asynchronous methods takes a last parameter as completion function callback and first parameter of the callback function is error. It is preferred to use asynchronous method instead of synchronous method as former never block the program execution where the latter one does.

- **Example:**



Create a text file named **input.txt** with the following content −

```
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

Let us create a js file named **main.js** with the following code −

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

// Asynchronous read
fs.readFile('input.txt', function (err, data) {
   if (err) {
      return console.error(err);
   }
   console.log("Asynchronous read: " + data.toString());
});

// Synchronous read
var data = fs.readFileSync('input.txt');
console.log("Synchronous read: " + data.toString());

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

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Synchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!

Program Ended
Asynchronous read: Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

34) Does Node.js supports Cryptography?

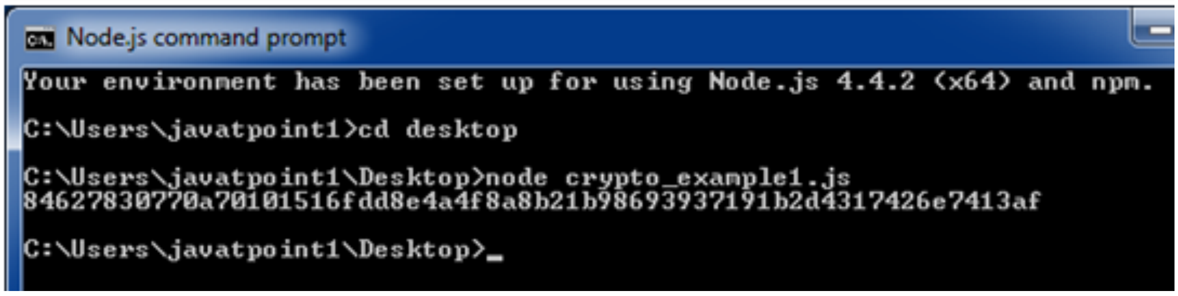
Yes, Node.js Crypto module supports cryptography. It provides cryptographic functionality that includes a set of wrappers for open SSL's hash HMAC, cipher, decipher, sign and verify functions. For example:

- Encryption Example using Hash and HMAC

```
const crypto = require('crypto');
const secret = 'abcdefg';
const hash = crypto.createHmac('sha256', secret)
                   .update('Welcome to JavaTpoint')
                   .digest('hex');
console.log(hash);
```

- Open Node.js command prompt and run the following code:

```
node crypto_example1.js
```



```
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 crypto_example1.js
84627830770a70101516fdd8e4a4f8a8b21b98693937191b2d4317426e7413af

C:\Users\javatpoint1\Desktop>_
```

35) Name some of the flags used in read/write operation on files?

flags for read/write operations are following:

Flag	Description
r	open file for reading. an exception occurs if the file does not exist.
r+	open file for reading and writing. an exception occurs if the file does not exist.
rs	open file for reading in synchronous mode.
rs+	open file for reading and writing, telling the os to open it synchronously. see notes for 'rs' about using this with caution.
w	open file for writing. the file is created (if it does not exist) or truncated (if it exists).
wx	like 'w' but fails if path exists.
w+	open file for reading and writing. the file is created (if it does not exist) or truncated (if it exists).
wx+	like 'w+' but fails if path exists.
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.

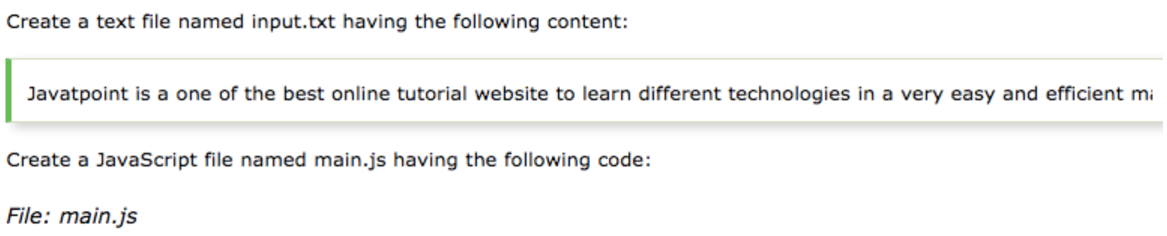
36) What are streams?

Streams are objects that let you read data from a source or write data to a destination in continuous fashion.

37) How many types of streams are present in Node?

In Node.js, there are four types of streams.

- **Readable** – Stream which is used for read operation.

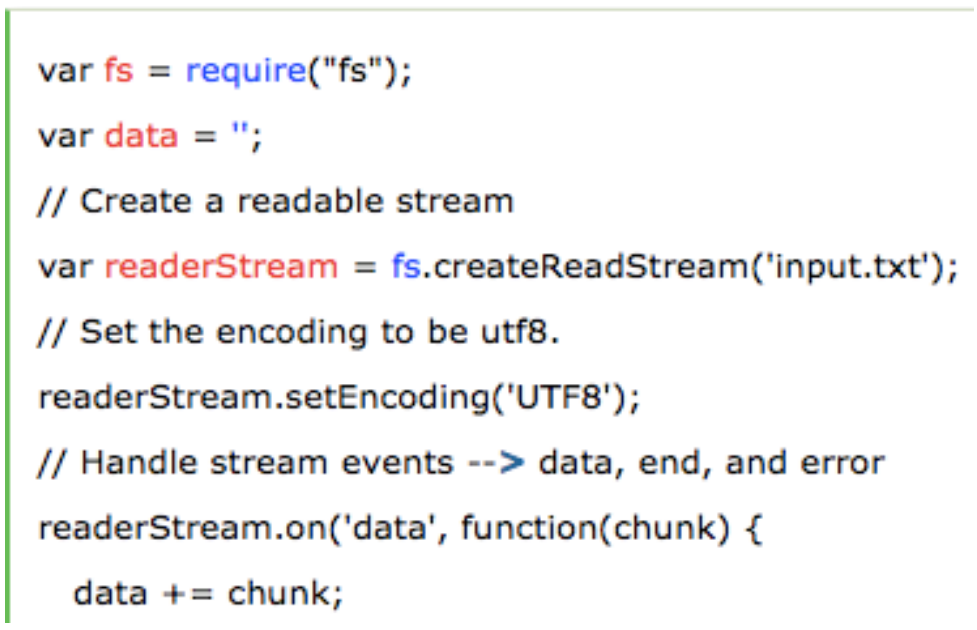


Create a text file named input.txt having the following content:

```
Javatpoint is a one of the best online tutorial website to learn different technologies in a very easy and efficient m
```

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

File: main.js

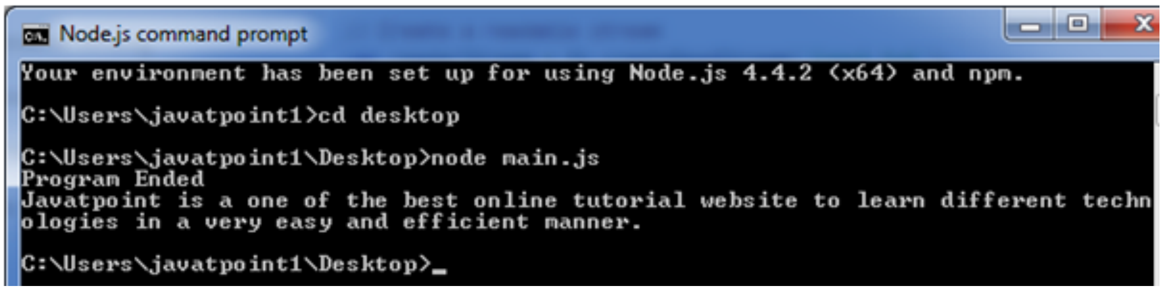


```
var fs = require("fs");
var data = '';
// Create a readable stream
var readerStream = fs.createReadStream('input.txt');
// Set the encoding to be utf8.
readerStream.setEncoding('UTF8');
// Handle stream events --> data, end, and error
readerStream.on('data', function(chunk) {
   data += chunk;
});
readerStream.on('end',function(){
   console.log(data);
});
readerStream.on('error', function(err){
   console.log(err.stack);
});
console.log("Program Ended");
```

Now, open the Node.js command prompt and run the main.js

```
node main.js
```

Output:



```
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 main.js
Program Ended
Javatpoint is a one of the best online tutorial website to learn different techn
ologies in a very easy and efficient manner.

C:\Users\javatpoint1\Desktop>_
```

- **Writable** – Stream which is used for write operation.

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

File: main.js

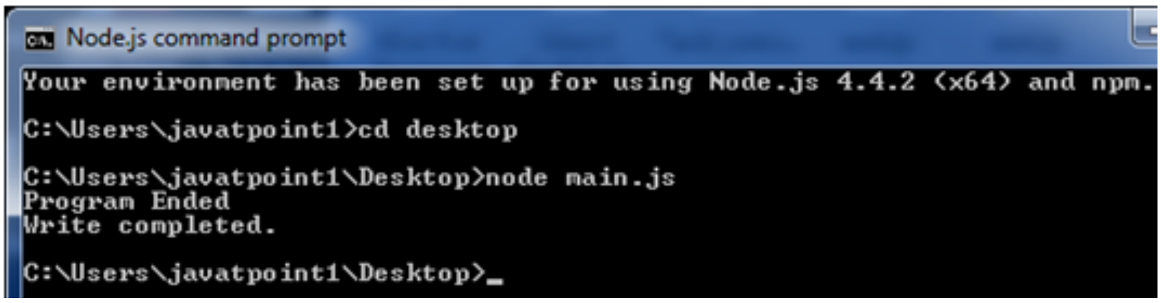
```
var fs = require("fs");
var data = 'A Solution of all Technology';
// Create a writable stream
var writerStream = fs.createWriteStream('output.txt');
// Write the data to stream with encoding to be utf8
writerStream.write(data,'UTF8');
```

```
// Mark the end of file
writerStream.end();
// Handle stream events --> finish, and error
writerStream.on('finish', function() {
    console.log("Write completed.");
});
writerStream.on('error', function(err){
   console.log(err.stack);
});
console.log("Program Ended");
```

Now open the Node.js command prompt and run the main.js

```
node main.js
```

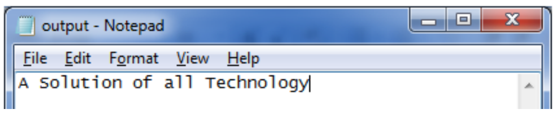
You will see the following result:



```
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 main.js
Program Ended
Write completed.

C:\Users\javatpoint1\Desktop>_
```

Now, you can see that a text file named "output.txt" is created where you had saved "input.txt" and "main.js" file. In my case, it is on desktop.

Open the "output.txt" and you will see the following content.



```
A Solution of all Technology
```

- **Duplex** – Stream which can be used for both read and write operation.
- **Transform** – A type of duplex stream where the output is computed based on input.

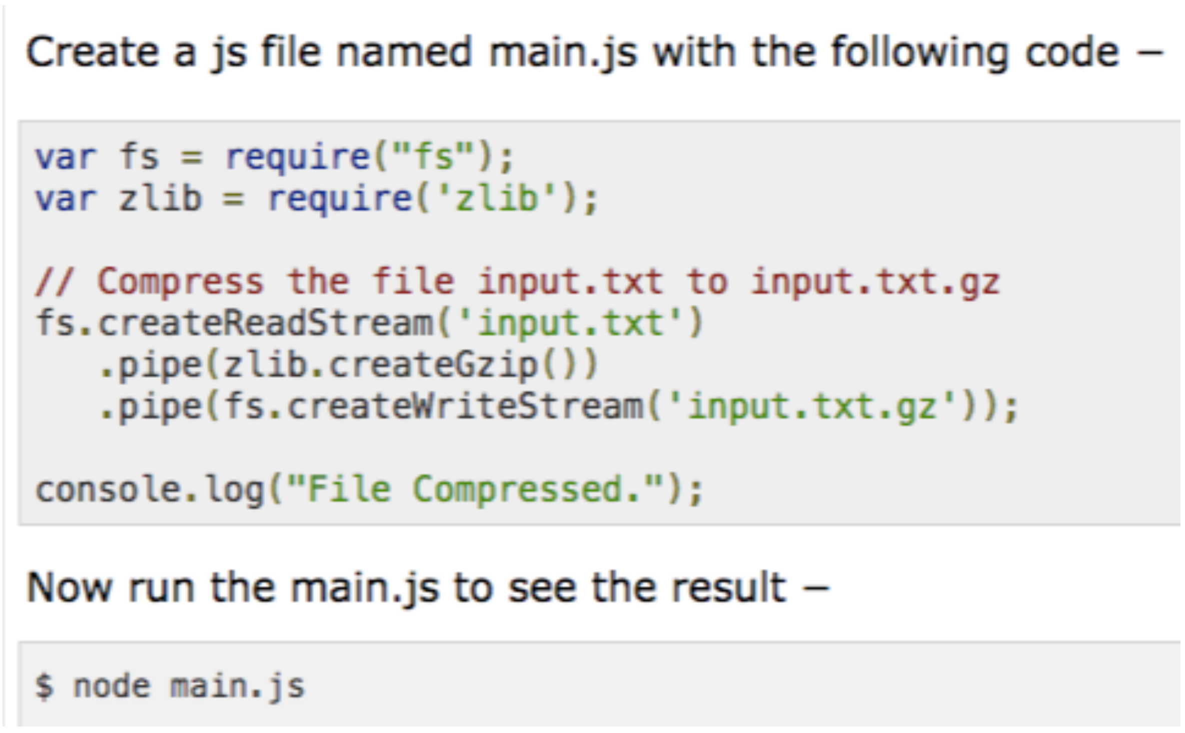
38) Name some of the events fired by streams?

Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times. For example, some of the commonly used events are:

- **data** – This event is fired when there is data is available to read.
- **end** – This event is fired when there is no more data 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.

39) What is Chaining in Node?

Chaining is a mechanism to connect output of one stream to another stream and create a chain of multiple stream operations. It is normally used with piping operations. Example:

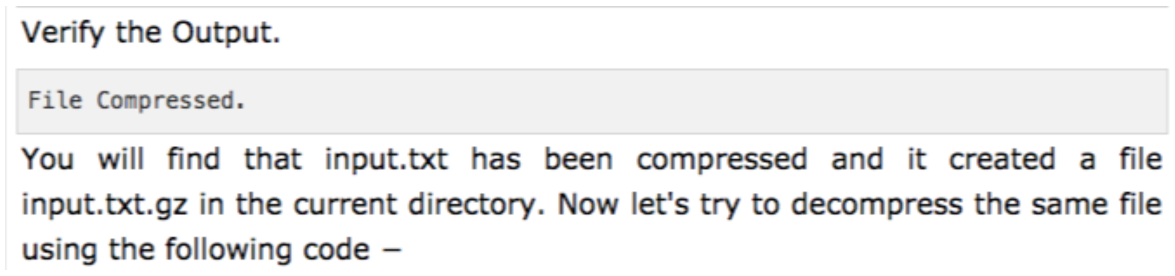


Create a js file named main.js with the following code –

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

// Compress the file input.txt to input.txt.gz
fs.createReadStream('input.txt')
   .pipe(zlib.createGzip())
   .pipe(fs.createWriteStream('input.txt.gz'));

console.log("File Compressed.");
```

Now run the main.js to see the result –

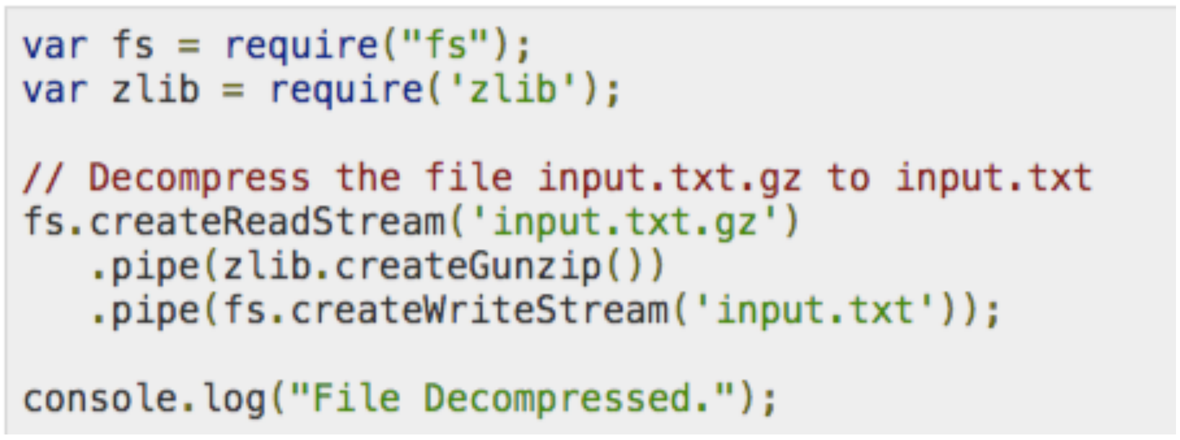
```
$ node main.js
```



Verify the Output.

```
File Compressed.
```

You will find that input.txt has been compressed and it created a file input.txt.gz in the current directory. Now let's try to decompress the same file using the following code –



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

// Decompress the file input.txt.gz to input.txt
fs.createReadStream('input.txt.gz')
   .pipe(zlib.createGunzip())
   .pipe(fs.createWriteStream('input.txt'));

console.log("File Decompressed.");
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
File Decompressed.
```

40) What is the Punycode in Node.js?

The Punycode is an encoding syntax which is used to convert Unicode (UTF-8) string of characters to ASCII string of characters. It is bundled with Node.js v0.6.2 and later versions. If you want to use it with other Node.js versions, then use npm to install Punycode module first. You have to used require ('Punycode') to access it. Syntax:

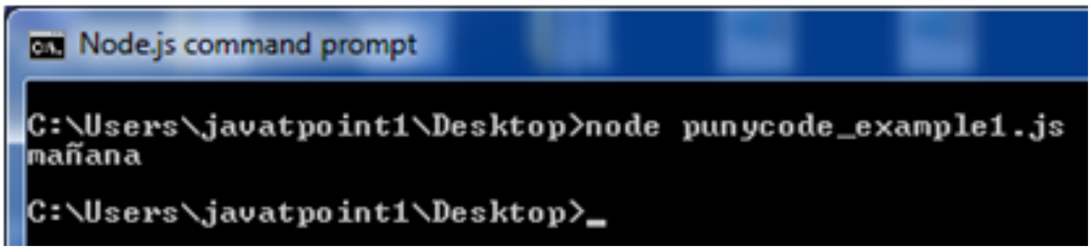
```
punycode = require('punycode');
```

- **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'));
```

Output:


Node.js command prompt

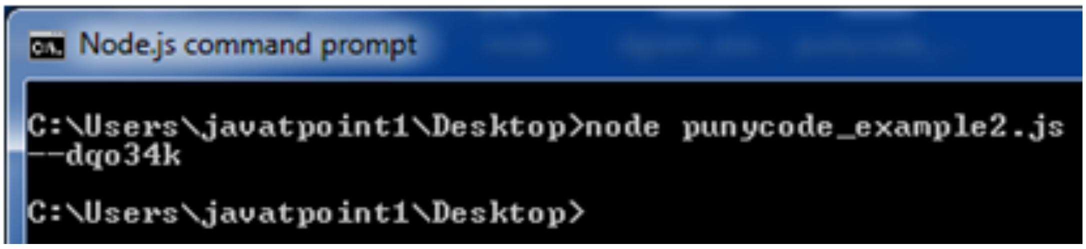
```
C:\Users\javatpoint1\Desktop>node punycode_example1.js
mañana

C:\Users\javatpoint1\Desktop>_
```

- **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('☃-⌘'));
```

Output:



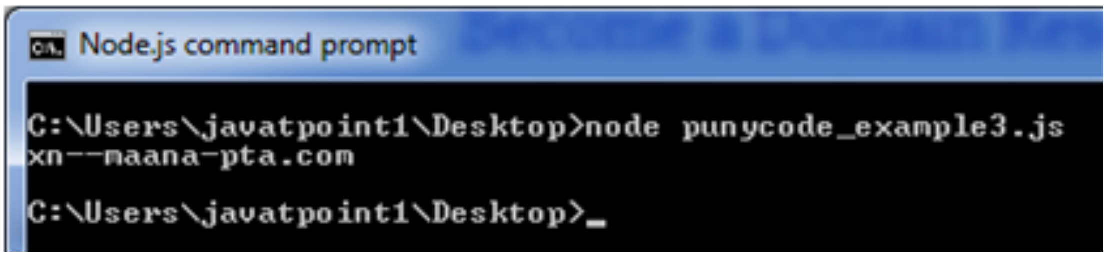
```
C:\Users\javatpoint1\Desktop>node punycode_example2.js
--dqo34k

C:\Users\javatpoint1\Desktop>
```

- **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'));
```

Output:



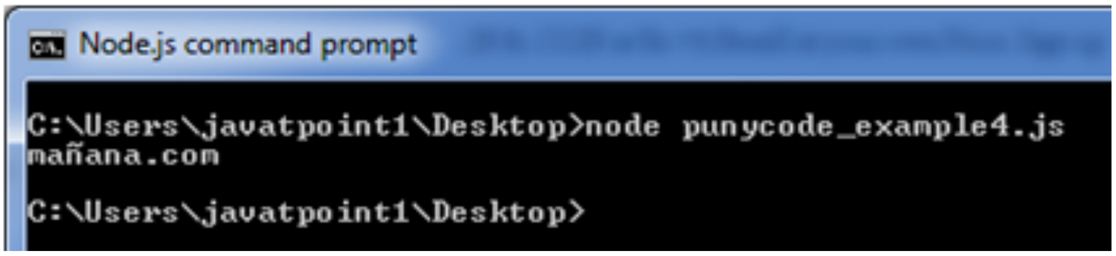
```
C:\Users\javatpoint1\Desktop>node punycode_example3.js
xn--maana-pta.com

C:\Users\javatpoint1\Desktop>_
```

- **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'));
```

Output:



```
C:\Users\javatpoint1\Desktop>node punycode_example4.js
mañana.com

C:\Users\javatpoint1\Desktop>
```

41) How will you open a file using Node?

Following is the syntax of the method to open a file in asynchronous mode:

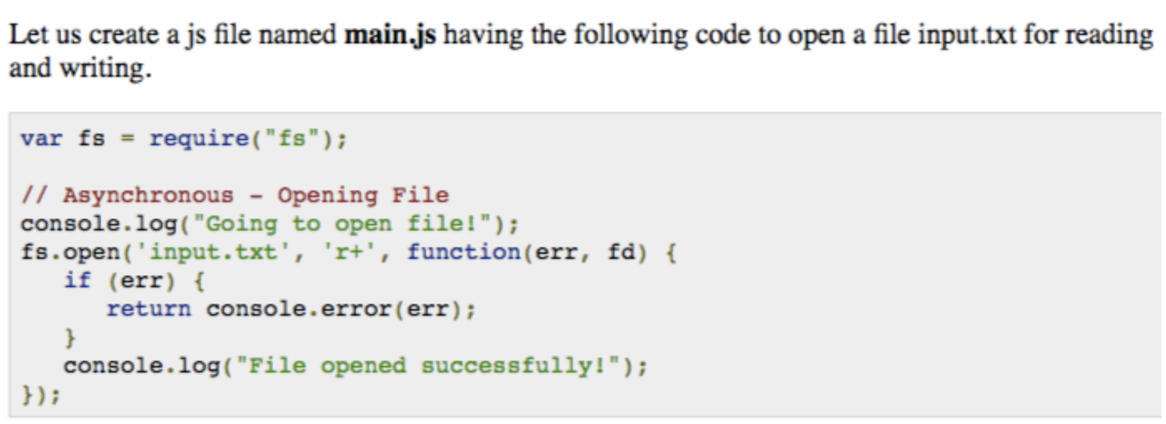
```
fs.open(path, flags[, mode], callback)
```

Parameters:

Here is the description of the parameters used:

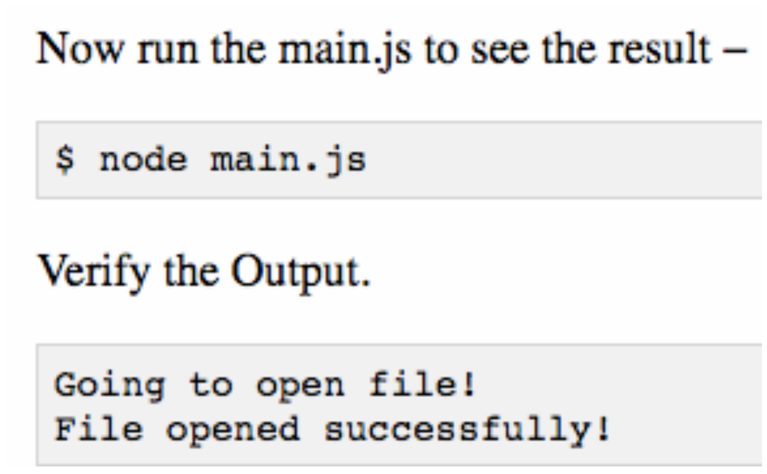
- **path** – This is string having file name including path.
- **flags** – Flag tells the behavior of the file to be opened. All possible values have been mentioned below.
- **mode** – This sets the file mode permissionandstickybits, but only if the file was created. It defaults to 0666, readable and writeable.
- **callback** – This is the callback function which gets two arguments err, fd.

Example:



Let us create a js file named **main.js** having the following code to open a file input.txt for reading and writing.

```
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 successfully!");
});
```



Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Going to open file!
File opened successfully!
```

42) How will you read a file using Node?

Following is the syntax of one of the methods to read from a file:

```
fs.read(fd, buffer, offset, length, position, callback)
```

This method will use file descriptor to read the file, if you want to read file using file name directly then you should use another method available.

Parameters:

Here is the description of the parameters used:

- **fd** – This is the file descriptor returned by file fs.open method.
- **buffer** – This is the buffer that the data will be written to.
- **offset** – This is the offset in the buffer to start writing at.
- **length** – This is an integer specifying the number of bytes to read.
- **position** – This is an integer specifying where to begin reading from in the file. If position is null, data will be read from the current file position.
- **callback** – This is the callback function which gets the three arguments, err, bytesRead, buffer.

Example:

Let us create a js file named **main.js** with the following code –

```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");
```

```
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes){
      if (err){
         console.log(err);
      }
      console.log(bytes + " bytes read");

      // Print only read bytes to avoid junk.
      if(bytes > 0){
         console.log(buf.slice(0, bytes).toString());
      }
   });
});
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Going to open an existing file
File opened successfully!
Going to read the file
97 bytes read
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!
```

43) How will you write a file using Node?

Following is the syntax of one of the methods to write into a file:

```
fs.writeFile(filename, data[, options], callback)
```

This method will over-write the file if file already exists. If you want to write into an existing file then you should use another method available.

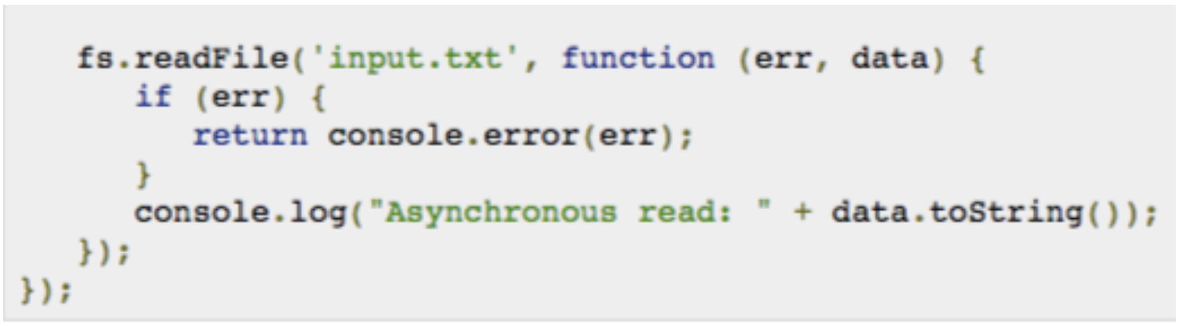
Parameters:

Here is the description of the parameters used:

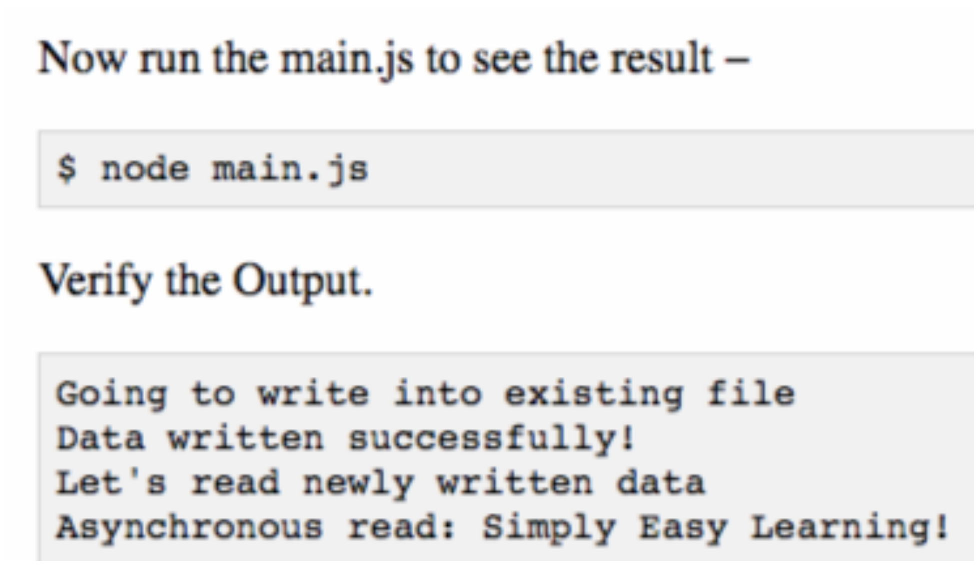
- **path** – This is string having file name including path.
- **data** – This is the String or Buffer to be written into the file.
- **options** – The third parameter is an object which will hold {encoding, mode, flag}. By default encoding is utf8, mode is octal value 0666 and flag is 'w'.
- **callback** – This is the callback function which gets a single parameter err and used to to return error in case of any writing error.

Example:

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

console.log("Going to write into existing file");
fs.writeFile('input.txt', 'Simply Easy Learning!', function(err) {
   if (err) {
      return console.error(err);
   }

   console.log("Data written successfully!");
   console.log("Let's read newly written data");
```



```
   fs.readFile('input.txt', function (err, data) {
      if (err) {
         return console.error(err);
      }
      console.log("Asynchronous read: " + data.toString());
   });
});
```



Now run the main.js to see the result –

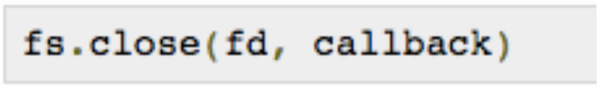
```
$ node main.js
```

Verify the Output.

```
Going to write into existing file
Data written successfully!
Let's read newly written data
Asynchronous read: Simply Easy Learning!
```

44) How will you close a file using Node?

Following is the syntax of one of the methods to close an opened file:



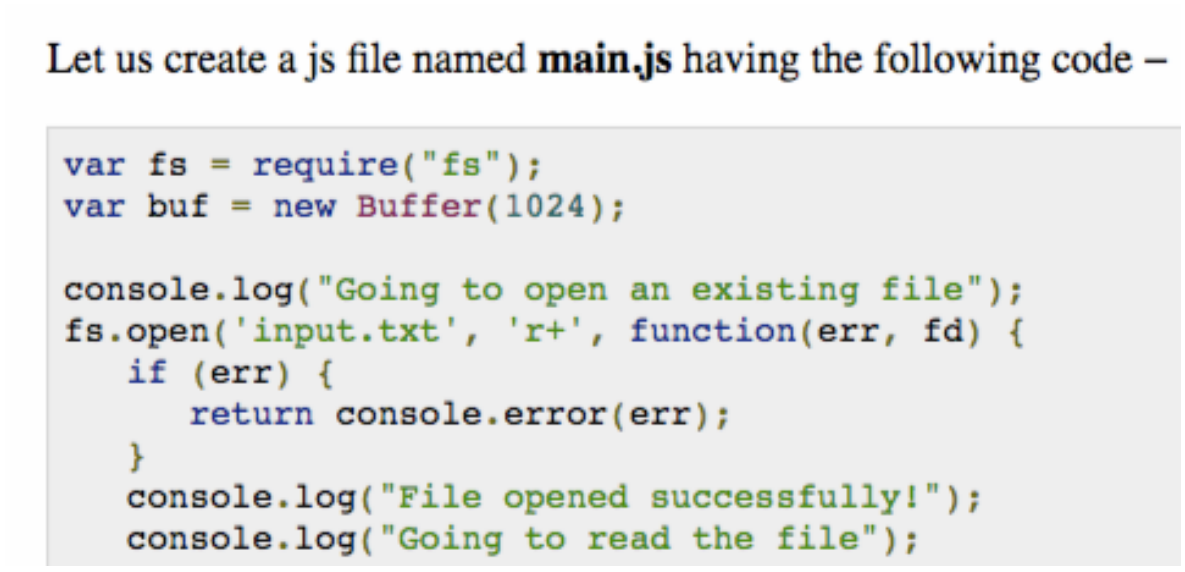
```
fs.close(fd, callback)
```

Parameters:

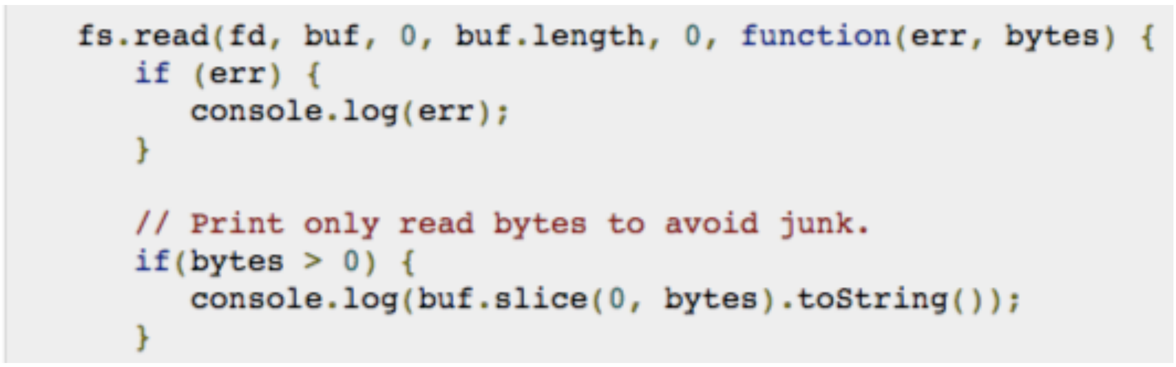
Here is the description of the parameters used:

- **fd** – This is the file descriptor returned by file fs.open method.
- **callback** – This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

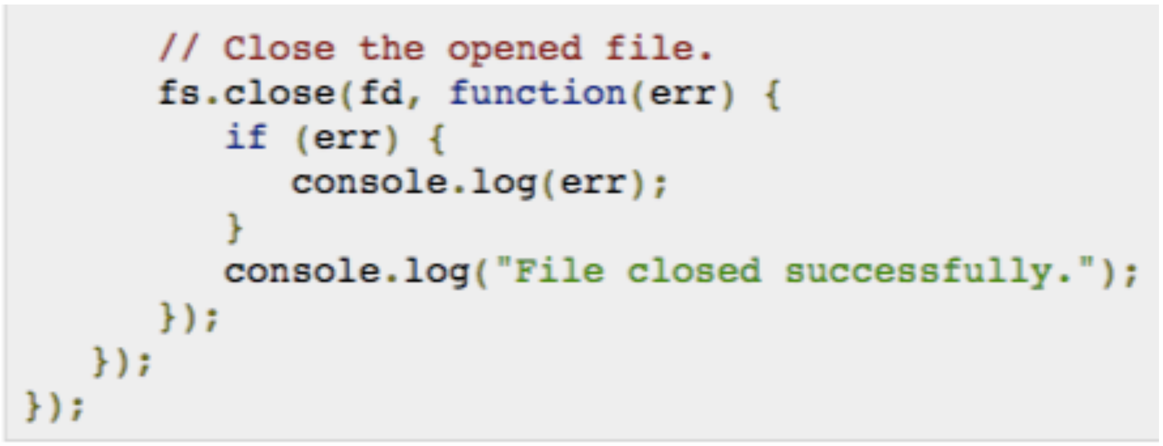
Example:


Let us create a js file named **main.js** having the following code –

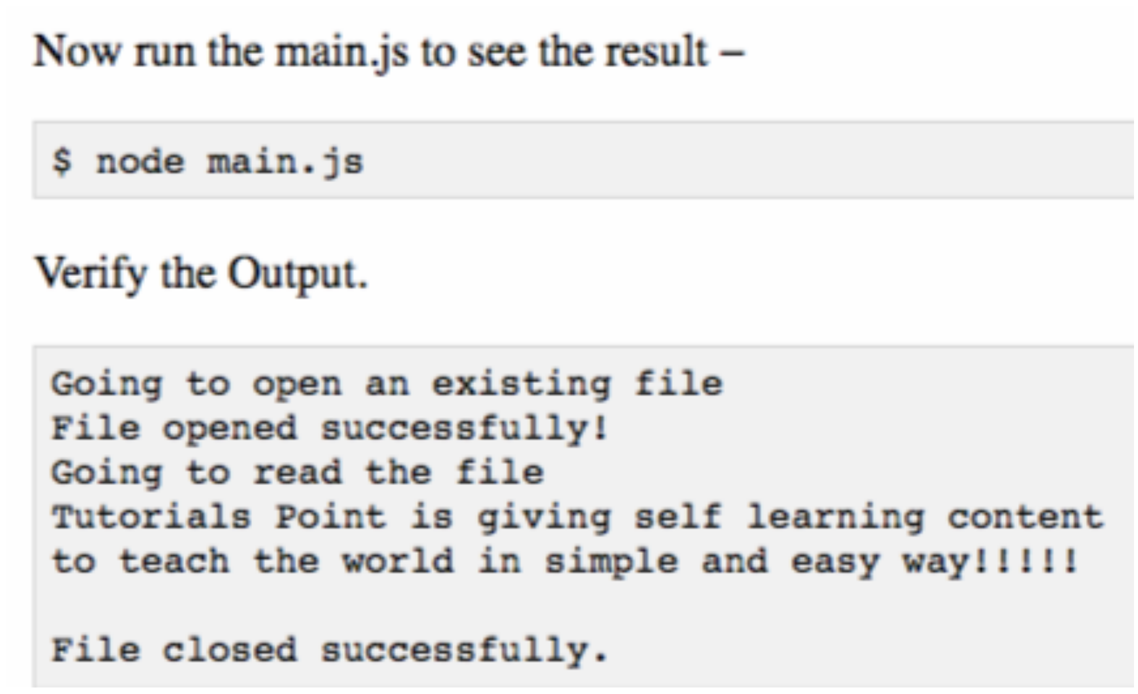
```
var fs = require("fs");
var buf = new Buffer(1024);

console.log("Going to open an existing file");
fs.open('input.txt', 'r+', function(err, fd) {
   if (err) {
      return console.error(err);
   }
   console.log("File opened successfully!");
   console.log("Going to read the file");
```



```
   fs.read(fd, buf, 0, buf.length, 0, function(err, bytes) {
      if (err) {
         console.log(err);
      }

      // Print only read bytes to avoid junk.
      if(bytes > 0) {
         console.log(buf.slice(0, bytes).toString());
      }
```



```
      // Close the opened file.
      fs.close(fd, function(err) {
         if (err) {
            console.log(err);
         }
         console.log("File closed successfully.");
      });
   });
});
```



Now run the main.js to see the result –

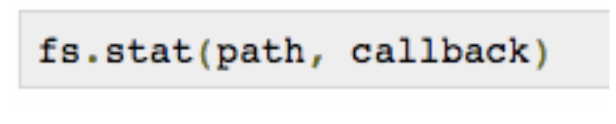
```
$ node main.js
```

Verify the Output.

```
Going to open an existing file
File opened successfully!
Going to read the file
Tutorials Point is giving self learning content
to teach the world in simple and easy way!!!!!

File closed successfully.
```

45) How will you get information about a file using Node?

Following is the syntax of the method to get the information about a file:



```
fs.stat(path, callback)
```

Parameters:

Here is the description of the parameters used:

- **path** – This is string having file name including path.
- **callback** – This is the callback function which gets two arguments err, stats where **stats** is an object of fs.Stats type which is printed below in the example.

Example:

Let us create a js file named **main.js** with the following code –

```
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("isDirectory ? " + stats.isDirectory());
});
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Going to get file info!
{
   dev: 1792,
   mode: 33188,
   nlink: 1,
   uid: 48,
   gid: 48,
   rdev: 0,
   blksize: 4096,
   ino: 4318127,
   size: 97,
   blocks: 8,
   atime: Sun Mar 22 2015 13:40:00 GMT-0500 (CDT),
   mtime: Sun Mar 22 2015 13:40:57 GMT-0500 (CDT),
   ctime: Sun Mar 22 2015 13:40:57 GMT-0500 (CDT)
}
Got file info successfully!
isFile ? true
isDirectory ? false
```

46) How will you truncate a file using Node?

Following is the syntax of the method to truncate an opened file:

```
fs.ftruncate(fd, len, callback)
```

Parameters:

Here is the description of the parameters used:

- **fd** – This is the file descriptor returned by file fs.open method.
- **len** – This is the length of the file after which file will be truncated.
- **callback** – This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

47) How will you delete a file using Node?

Following is the syntax of the method to delete a file:

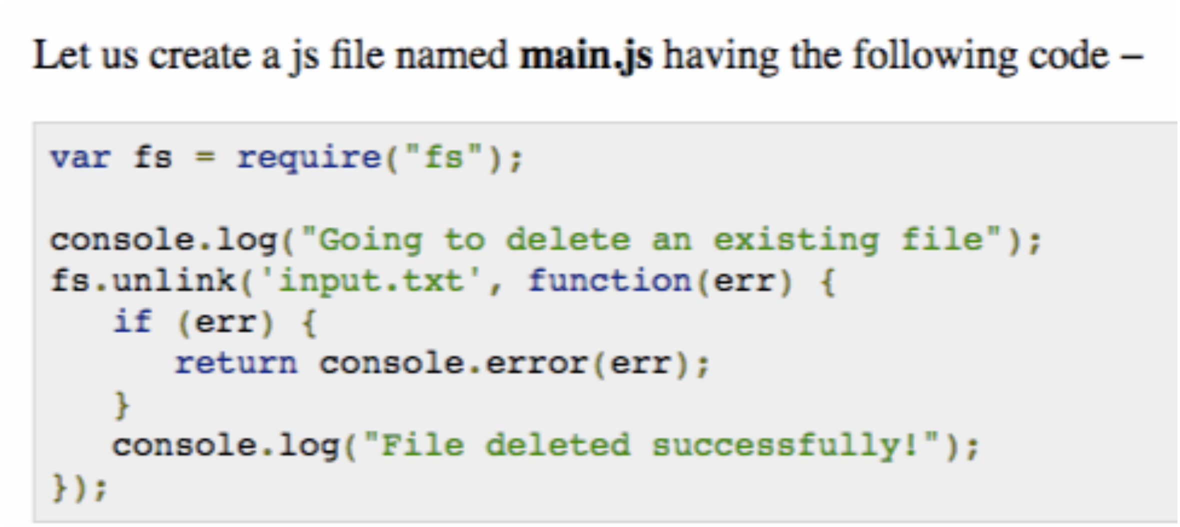
```
fs.unlink(path, callback)
```

Parameters:

Here is the description of the parameters used:

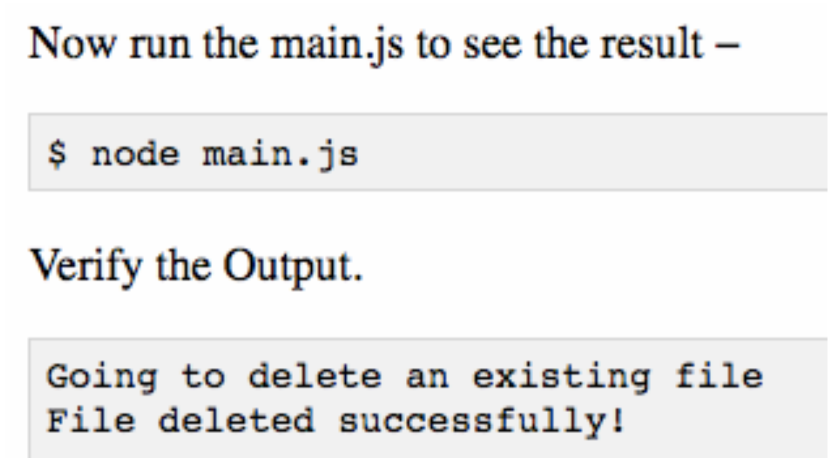
- **path** – This is the file name including path.
- **callback** – This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

Example:



Let us create a js file named **main.js** having the following code –

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

console.log("Going to delete an existing file");
fs.unlink('input.txt', function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("File deleted successfully!");
});
```



Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Going to delete an existing file
File deleted successfully!
```

48) How will you create a directory?

Following is the syntax of the method to create a directory:

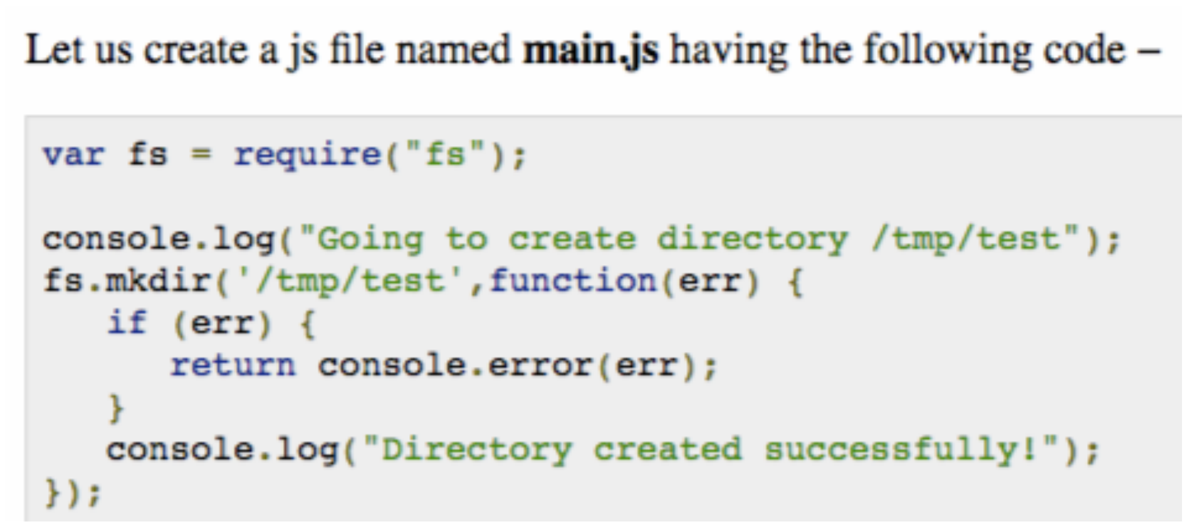
```
fs.mkdir(path[, mode], callback)
```

Parameters:

Here is the description of the parameters used:

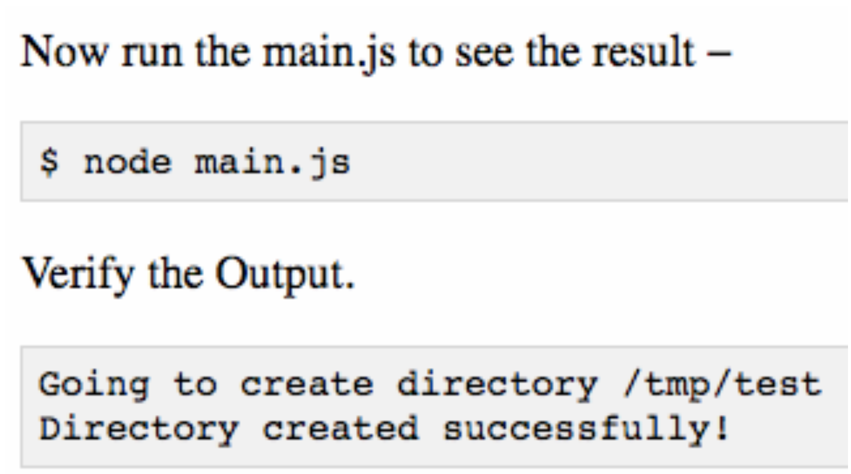
- **path** – This is the directory name including path.
- **mode** – This is the directory permission to be set. Defaults to 0777.
- **callback** – This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

Example:



Let us create a js file named **main.js** having the following code –

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

console.log("Going to create directory /tmp/test");
fs.mkdir('/tmp/test',function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Directory created successfully!");
});
```


Now run the main.js to see the result –

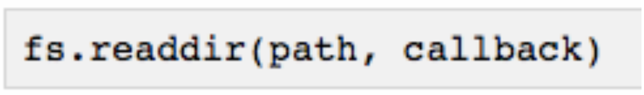
```
$ node main.js
```

Verify the Output.

```
Going to create directory /tmp/test
Directory created successfully!
```

49) How will you read a directory?

Following is the syntax of the method to read a directory:

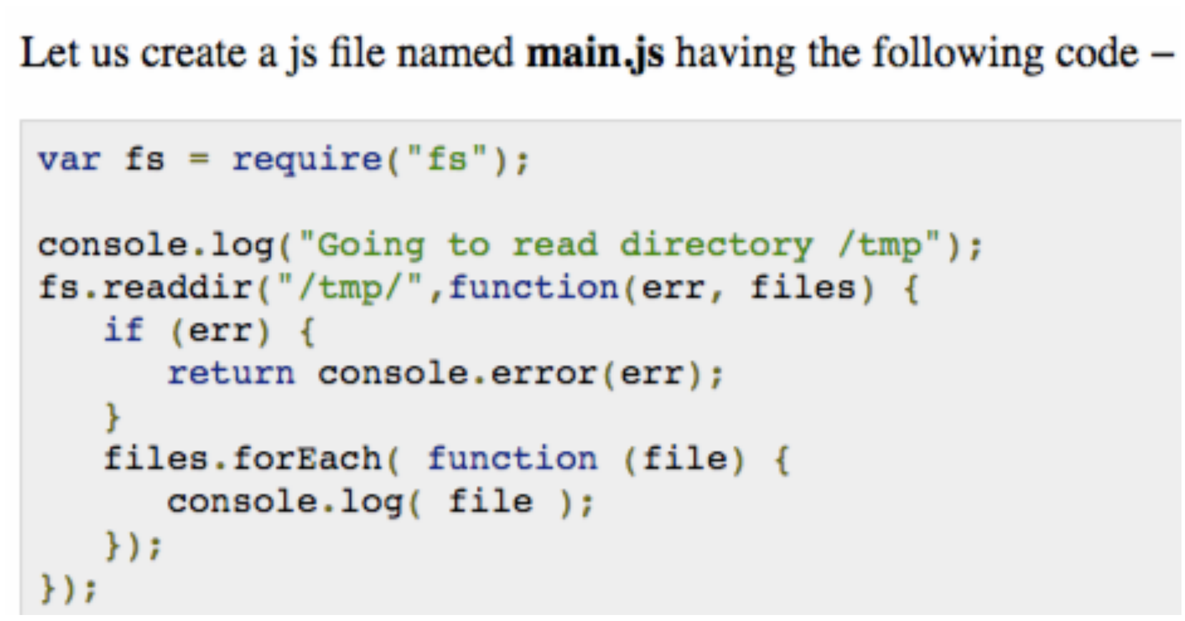

```
fs.readdir(path, callback)
```

Parameters:

Here is the description of the parameters used:

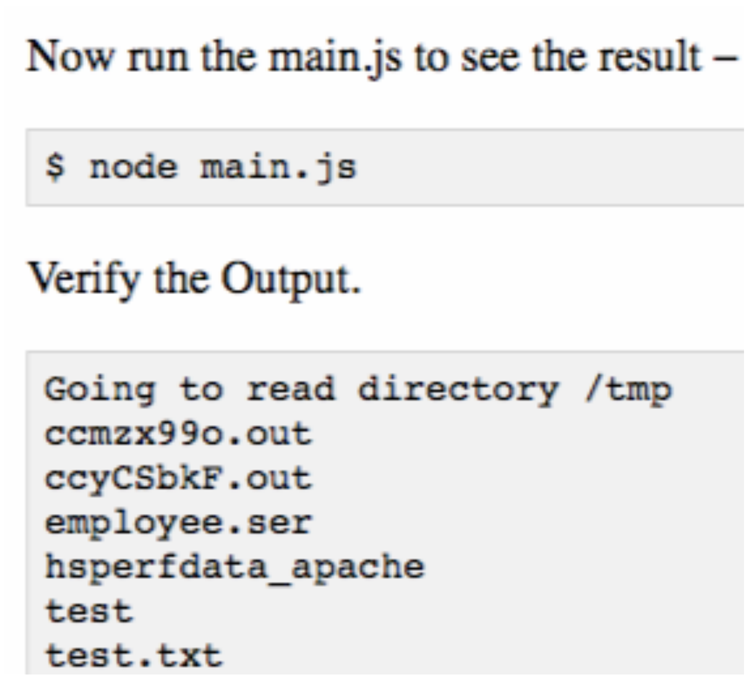
- **path** – This is the directory name including path.
- **callback** – This is the callback function which gets two arguments err, files where files is an array of the names of the files in the directory excluding '.' and '..'.

Example:



Let us create a js file named **main.js** having the following code –

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

console.log("Going to read directory /tmp");
fs.readdir("/tmp/",function(err, files) {
   if (err) {
      return console.error(err);
   }
   files.forEach( function (file) {
      console.log( file );
   });
});
```



Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output.

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test
test.txt
```

50) How will you delete a directory?

Following is the syntax of the method to remove a directory:

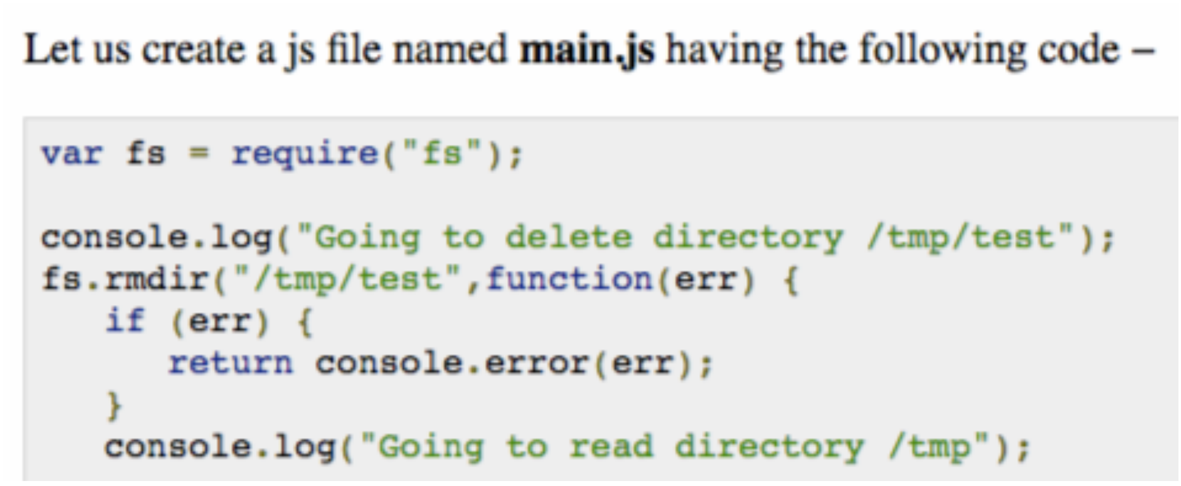
```
fs.rmdir(path, callback)
```

Parameters:

Here is the description of the parameters used:

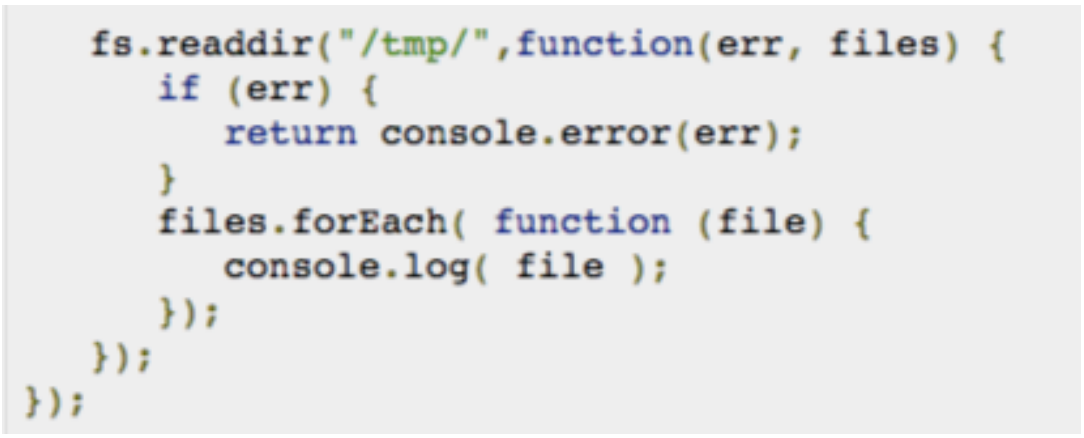
- **path** – This is the directory name including path.
- **callback** – This is the callback function which gets no arguments other than a possible exception are given to the completion callback.

Example:

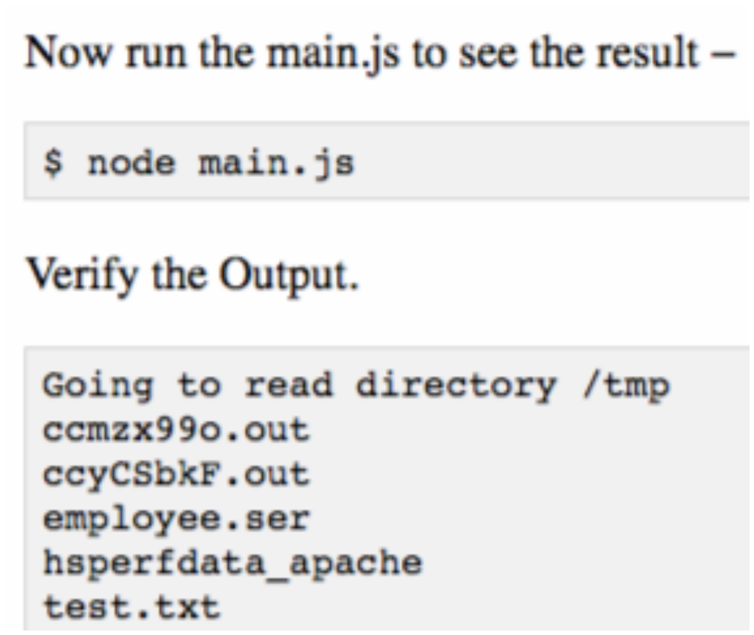


Let us create a js file named **main.js** having the following code –

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

console.log("Going to delete directory /tmp/test");
fs.rmdir("/tmp/test",function(err) {
   if (err) {
      return console.error(err);
   }
   console.log("Going to read directory /tmp");
```



```
   fs.readdir("/tmp/",function(err, files) {
      if (err) {
         return console.error(err);
      }
      files.forEach( function (file) {
         console.log( file );
      });
   });
});
```


Now run the main.js to see the result –

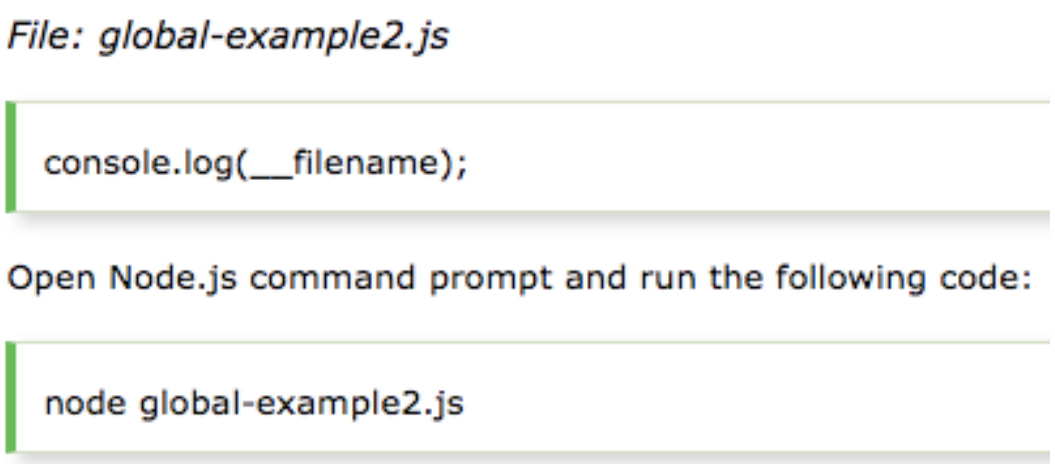
```
$ node main.js
```

Verify the Output.

```
Going to read directory /tmp
ccmzx99o.out
ccyCSbkF.out
employee.ser
hsperfdata_apache
test.txt
```

51) What is the purpose of __filename variable?

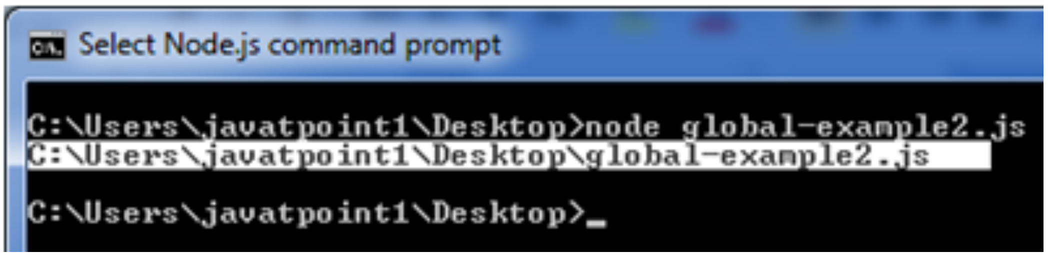
The __filename represents the filename of the code being executed. This is the resolved absolute path of this code file. For a main program this is not necessarily the same filename used in the command line. The value inside a module is the path to that module file.


File: global-example2.js

```
console.log(__filename);
```

Open Node.js command prompt and run the following code:

```
node global-example2.js
```



```
C:\Users\javatpoint1\Desktop>node global-example2.js
C:\Users\javatpoint1\Desktop\global-example2.js

C:\Users\javatpoint1\Desktop>_
```

52) What is the purpose of __dirname variable?

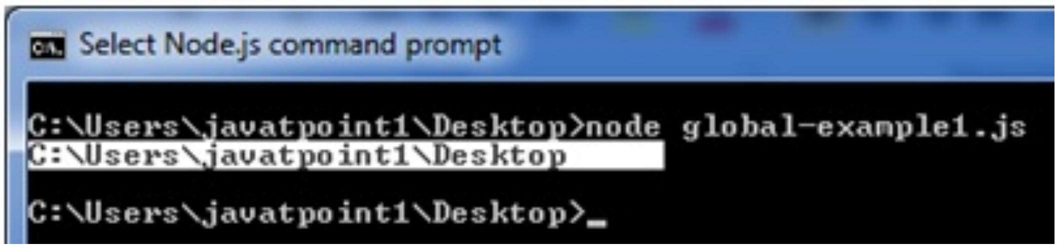
The __dirname represents the name of the directory that the currently executing script resides in.

File: global-example1.js

```
console.log(__dirname);
```

Open Node.js command prompt and run the following code:

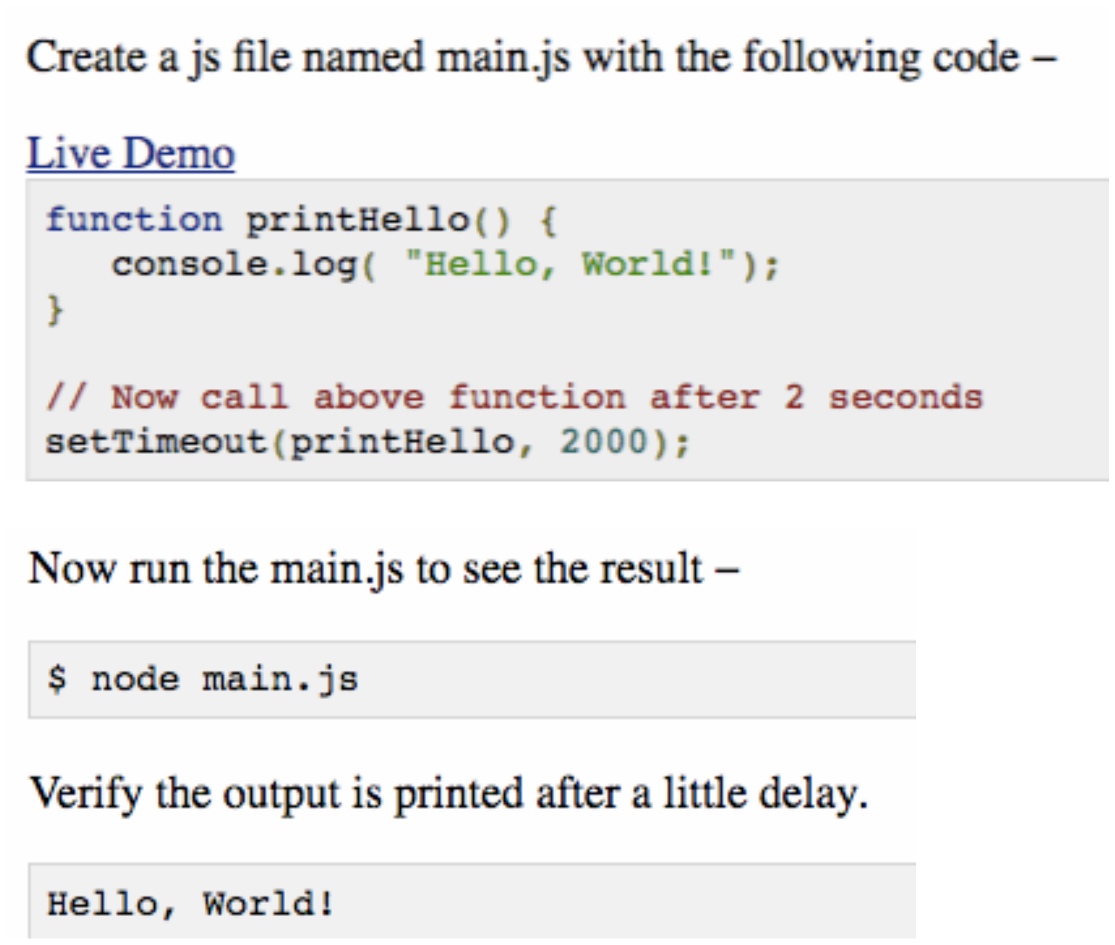
```
node global-example1.js
```



```
C:\Users\javatpoint1\Desktop>node global-example1.js
C:\Users\javatpoint1\Desktop

C:\Users\javatpoint1\Desktop>_
```

53) What is the purpose of setTimeout function?

The setTimeoutcb, ms global function is used to run callback cb after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days. This function returns an opaque value that represents the timer which can be used to clear the timer.



Create a js file named main.js with the following code –

Live Demo

```
function printHello() {
   console.log( "Hello, World!");
}

// Now call above function after 2 seconds
setTimeout(printHello, 2000);
```

Now run the main.js to see the result –

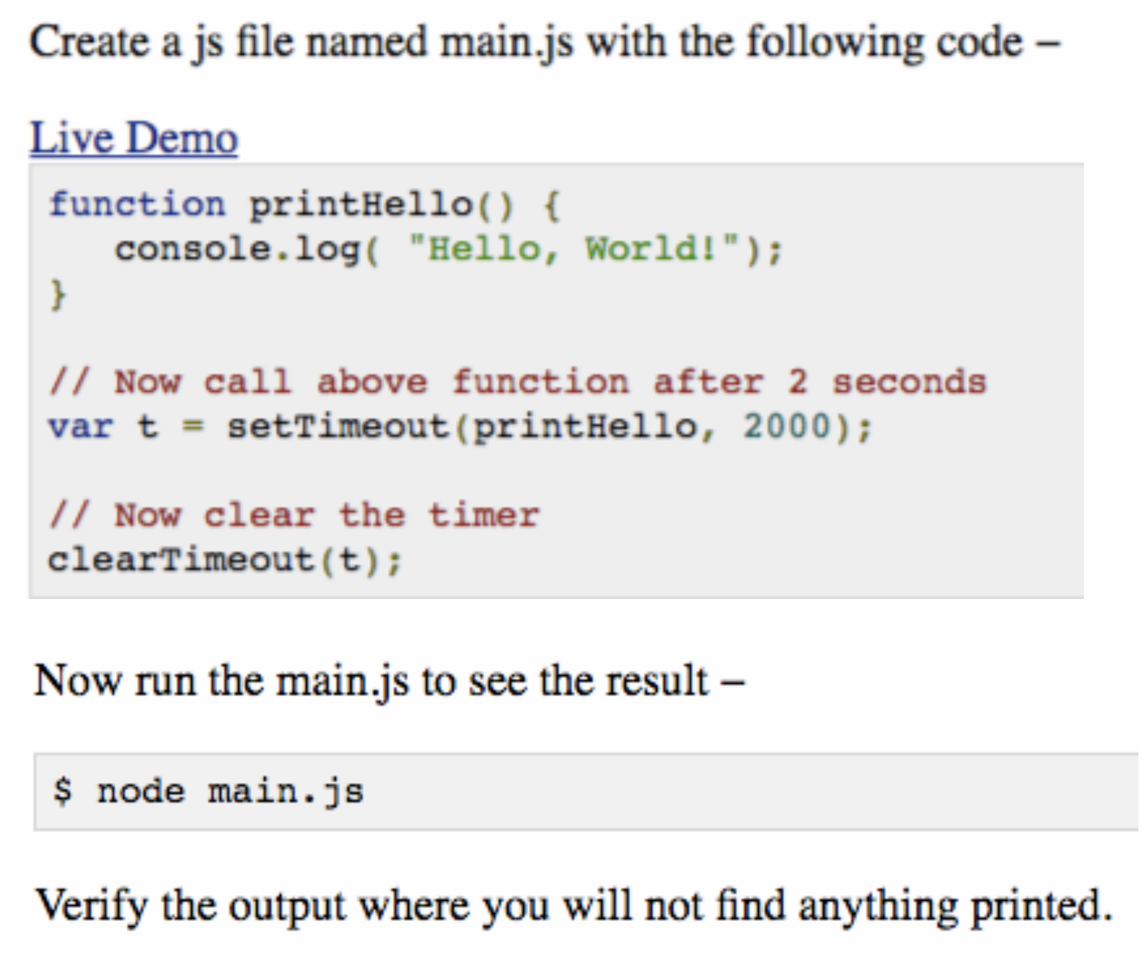
```
$ node main.js
```

Verify the output is printed after a little delay.

```
Hello, World!
```

54) What is the purpose of clearTimeout function?

The clearTimeoutt global function is used to stop a timer that was previously created with setTimeout. Here t is the timer returned by setTimeout function.



Create a js file named main.js with the following code –

Live Demo

```
function printHello() {
   console.log( "Hello, World!");
}

// Now call above function after 2 seconds
var t = setTimeout(printHello, 2000);

// Now clear the timer
clearTimeout(t);
```

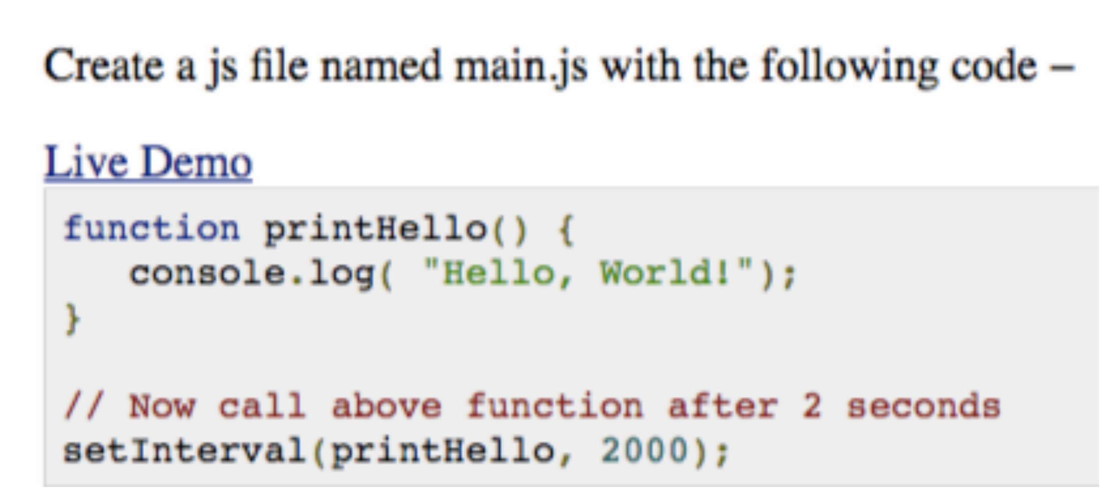
Now run the main.js to see the result –

```
$ node main.js
```

Verify the output where you will not find anything printed.

55) What is the purpose of setInterval function?

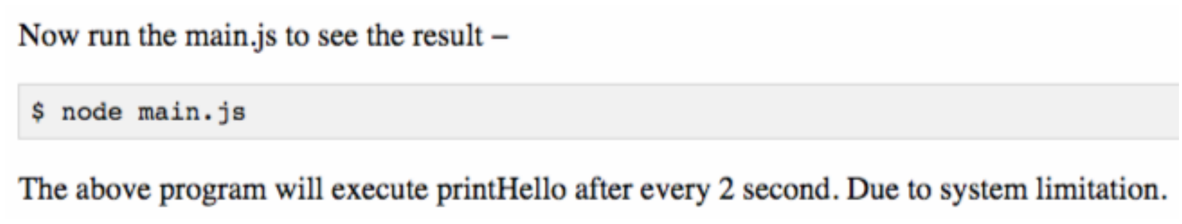
The setIntervalcb, ms global function is used to run callback cb repeatedly after at least ms milliseconds. The actual delay depends on external factors like OS timer granularity and system load. A timer cannot span more than 24.8 days. This function returns an opaque value that represents the timer which can be used to clear the timer using the function clearIntervalt.



Create a js file named main.js with the following code –

Live Demo

```
function printHello() {
   console.log( "Hello, World!");
}

// Now call above function after 2 seconds
setInterval(printHello, 2000);
```



Now run the main.js to see the result –

```
$ node main.js
```

The above program will execute printHello after every 2 second. Due to system limitation.

56) What is the purpose of console object?

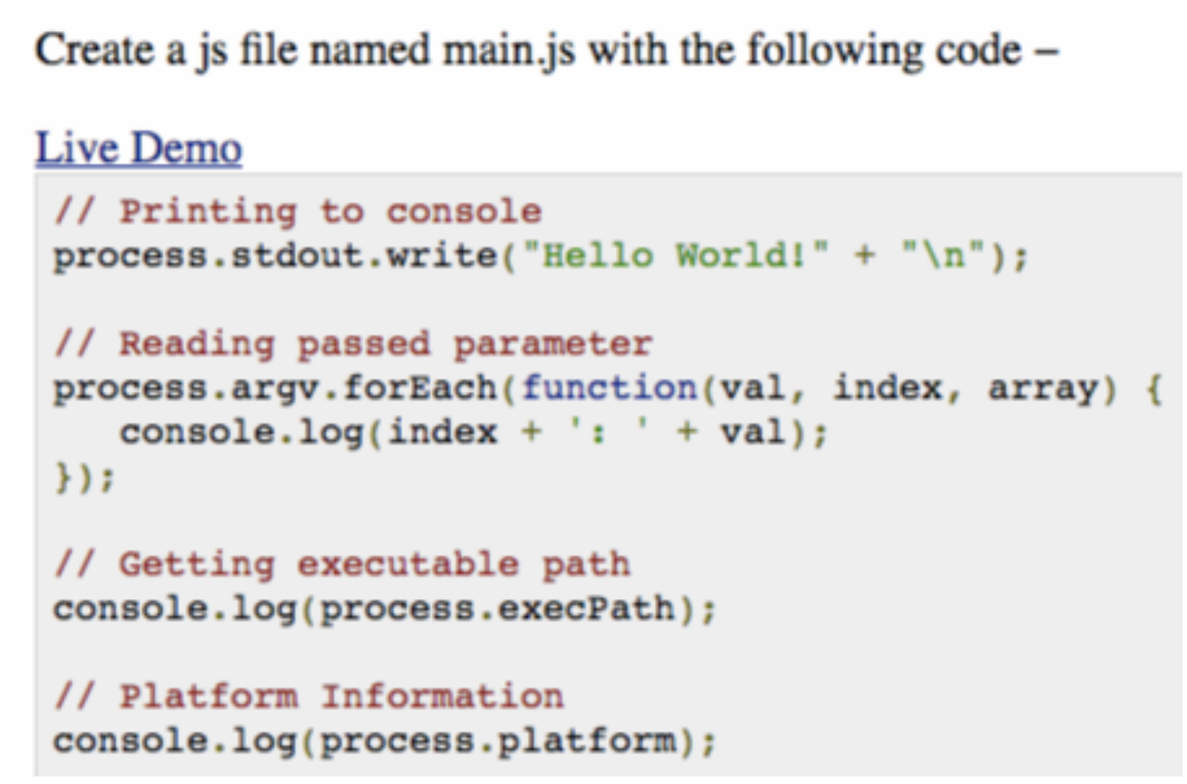
console object is used to Used to print information on stdout and stderr.

57) What is the purpose of process object?

process object is used to get information on current process. Provides multiple events related to process activities.

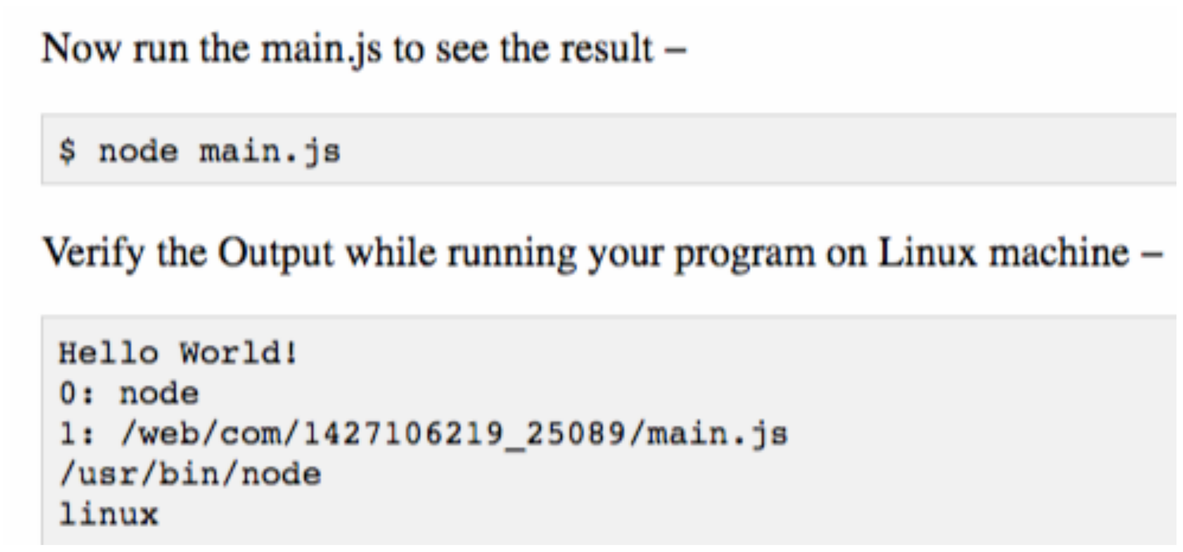
1. Process Properties

- **Example-1:**


Create a js file named main.js with the following code –

Live Demo

```
// Printing to console
process.stdout.write("Hello World!" + "\n");

// Reading passed parameter
process.argv.forEach(function(val, index, array) {
   console.log(index + ': ' + val);
});

// Getting executable path
console.log(process.execPath);

// Platform Information
console.log(process.platform);
```


Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output while running your program on Linux machine –

```
Hello World!
0: node
1: /web/com/1427106219_25089/main.js
/usr/bin/node
linux
```

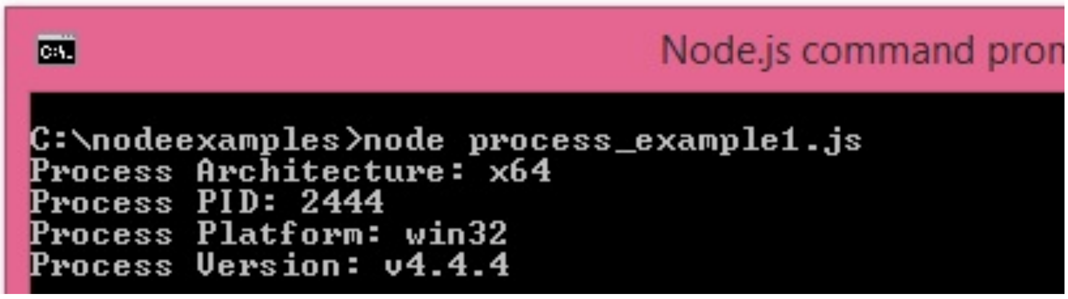
- **Example-2:**

File: process_example1.js

```
console.log(`Process Architecture: ${process.arch}`);
console.log(`Process PID: ${process.pid}`);
console.log(`Process Platform: ${process.platform}`);
console.log(`Process Version: ${process.version}`);
```

Open Node.js command prompt and run the following code:

```
node process_example1.js
```



```
C:\nodeexamples>node process_example1.js
Process Architecture: x64
Process PID: 2444
Process Platform: win32
Process Version: v4.4.4
```

2. Process Functions

- **Example-1:**

Create a js file named main.js with the following code –

Live Demo

```
// Print the current directory
console.log('Current directory: ' + process.cwd());

// Print the process version
console.log('Current version: ' + process.version);

// Print the memory usage
console.log(process.memoryUsage());
```

Now run the main.js to see the result –

```
$ node main.js
```

Verify the Output while running your program on Linux machine –

```
Current directory: /web/com/1427106219_25089
Current version: v0.10.33
{ rss: 11505664, heapTotal: 4083456, heapUsed: 2157704 }
```

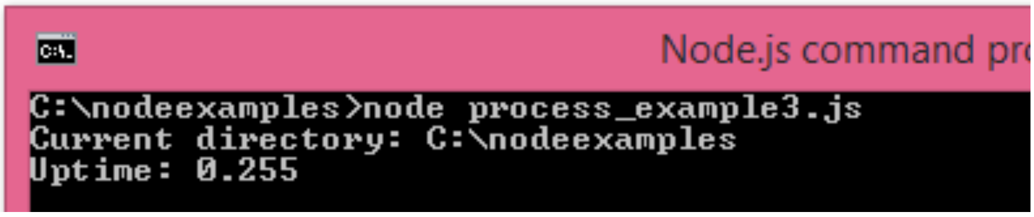
- **Example-2:**

File: process_example3.js

```
console.log(`Current directory: ${process.cwd()}`);
console.log(`Uptime: ${process.uptime()}`);
```

Open Node.js command prompt and run the following code:

```
node process_example3.js
```



```
C:\nodeexamples>node process_example3.js
Current directory: C:\nodeexamples
Uptime: 0.255
```

58) Node.js vs Java?

Index	Node.js	Java
1.	Node.js is single-threaded.	Java is multi-threaded
2.	It has asynchronous I/O.	It has synchronous I/O.
3.	Node.js is faster than Java because of its asynchronous and non-blocking nature.	Java is synchronous in nature so it is slower than Node.js.

59) Node.js advantages over Java?

- Node.js shows extremely good performance. It is almost 20% faster than Java.
- Node.js has active and vibrant community, with lots of code shared via github, etc.
- Node.js has growing number of good npm libraries.
- Node.js has an asynchronous IO which is the future for concurrency and scalability.
- Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. So it is preferred for video uploading.
- A web server written in Node.js will be faster than apache.