

5.1 Concept, Working and Features

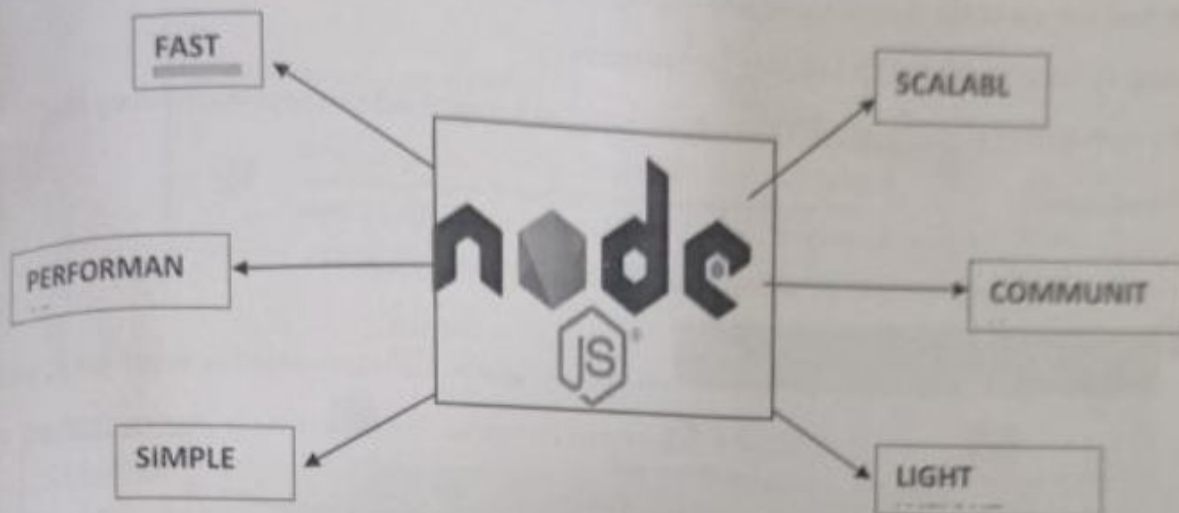
- Node.js is a server-side platform. It is built on Google Chrome's JavaScript Engine (V8 Engine).
 - It was developed by Ryan Dahl in 2009 and its latest version is 16.13.1 (includes npm 8.1.2)
 - The definition of Node.js given by its official documentation is as follows –
 - "Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices."
 - Node.js is an open source, cross-platform runtime environment.
 - It is used for developing server-side and networking applications.
 - Node.js applications are written in JavaScript.
 - It can be run inside the Node.js runtime on OS X, Microsoft Windows, and Linux.
 - Node.js also provides a huge library of different JavaScript modules
 - Such libraries make the development of web applications easier.
- Node.js = Runtime Environment + JavaScript Library

5.1.1 Features of Node.js

Following are some of the important features of Node.js:

- **Asynchronous** – All APIs of Node.js library is asynchronous. This means that they are non-blocking. Server based on Node.js never waits for an API to return data. The server moves to the next API after calling it.
- **Event Driven** The event notification of Node.js helps the server to get a response from the previous API call.
- **Very Fast** – Since it is built on Google Chrome's V8 JavaScript Engine, Node.js library is very fast in code execution.
- **Single Threaded** – Node.js uses a single threaded model with event looping. Event mechanism helps the server to respond in a non-blocking way.

- **Highly Scalable**- Node.js uses a single threaded program and the same program can provide service to many requests making it highly scalable
- **No Buffering** - Applications of Node.js never buffer any data. They just give data as output in chunks.
- **License** - Node.js is released under the MIT license.



Following are the areas or applications where Node.js can preferred to be used:

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real-time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Where Not to Use Node.js?

It is not advisable to use Node.js for CPU intensive applications.

5.1.2 Downloading Node.js

- Node.js can be downloaded from its official website <https://nodejs.org/en/download/>.
- On this site click the **Windows Installer** button to download the latest default version.
- The Node.js installer also includes the NPM. NPM is the **package manager for the Node JavaScript platform**. It puts modules in the location from where the node can

find them, and use them whenever required. So in general, it is used to publish, discover, install, and develop node programs.

5.2 Setting up Node.js server

5.2.1 Installing on window

Below are the steps to download and install Node.js in Windows:

Step 1) Download Node.js Installer for Windows

Go to the site <https://nodejs.org/en/download/> and download the necessary binary files.

Downloads

Latest LTS Version: 16.13.1 (includes npm 8.3.2)

Download the Node.js source code or a pre-built installer for your platform, and start developing today.

The screenshot shows the Node.js download page. The 'LTS' section is highlighted with a dark header. Below it, there are three main download categories: Windows Installer, macOS Installer, and Source Code. Under Windows Installer, there are links for Windows Installer (.msi), Windows Binary (.zip), macOS Installer (.pkg), macOS Binary (.tar.gz), Linux Binaries (x64), Linux Binaries (ARM), and Source Code. A table below these links lists various binary options for Windows, macOS, and Linux. An arrow points from a text box on the right to the '32-bit' option in the Windows section of the table.

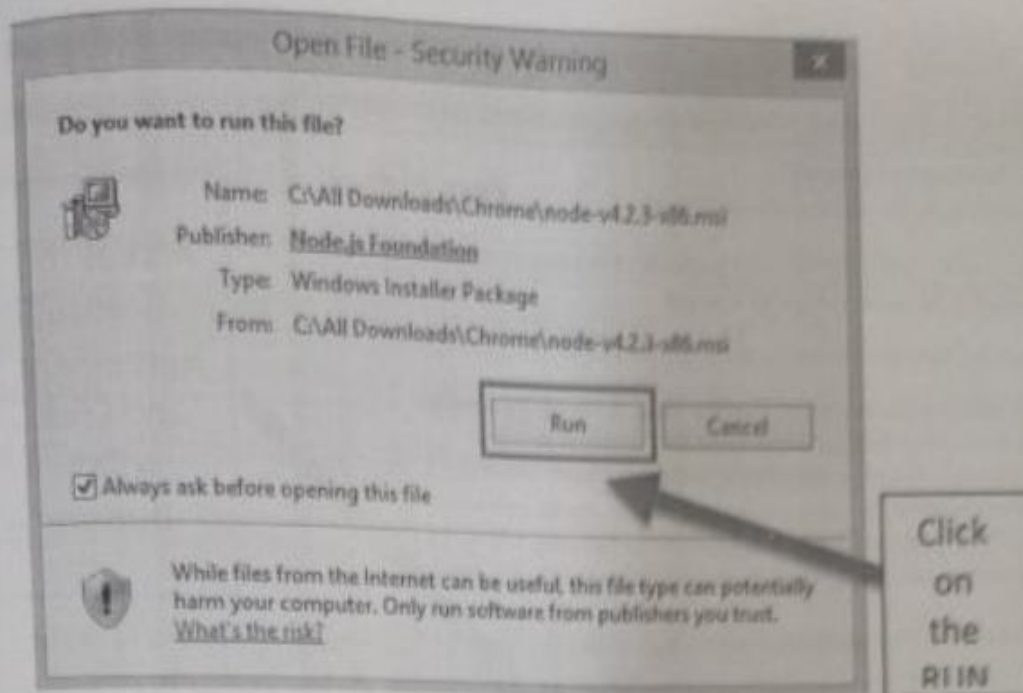
Platform	Architecture	Download Link
Windows	32-bit	node-v16.13.1-win32-x86.msi
	64-bit	node-v16.13.1-win64-x64.msi
macOS	64-bit	node-v16.13.1-darwin-x64.pkg
	ARM64	node-v16.13.1-darwin-arm64.pkg
Linux	x64	node-v16.13.1-linux-x64.tar.gz
	ARM64	node-v16.13.1-linux-arm64.tar.gz

Select 32
/64
bit.msi
s per
your
machine

Step 2) Run the installation

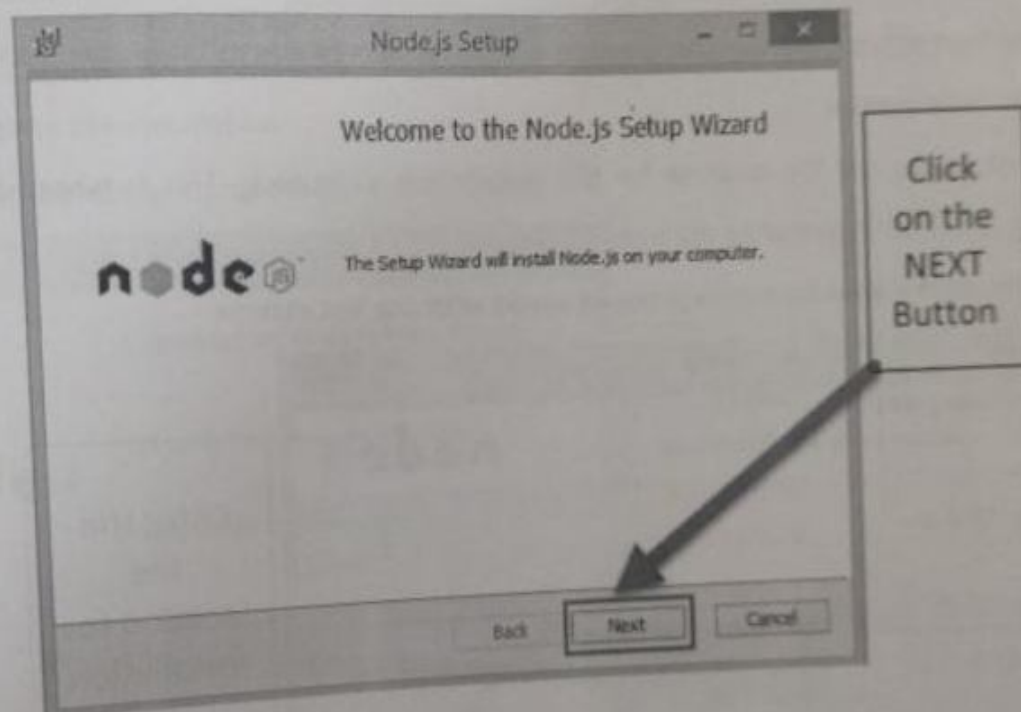
Double click on the downloaded .msi file to start the installation.

In the first screen ,click the Run button to begin the installation.



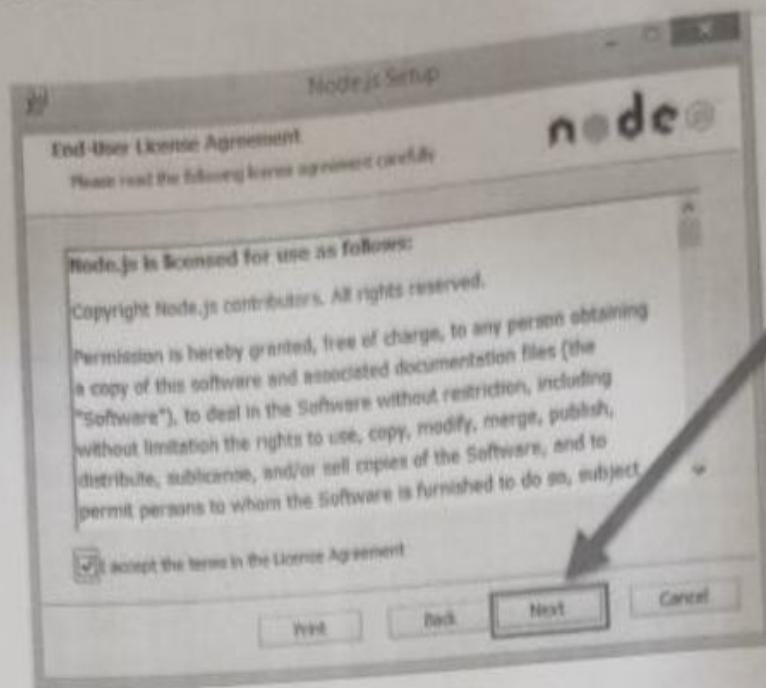
Step 3) Continue with the installation steps

In the next screen, click the "Next" button to continue the installation



Step 4) Accept the terms and conditions

In the next screen, Accept the license agreement and click on the Next button.

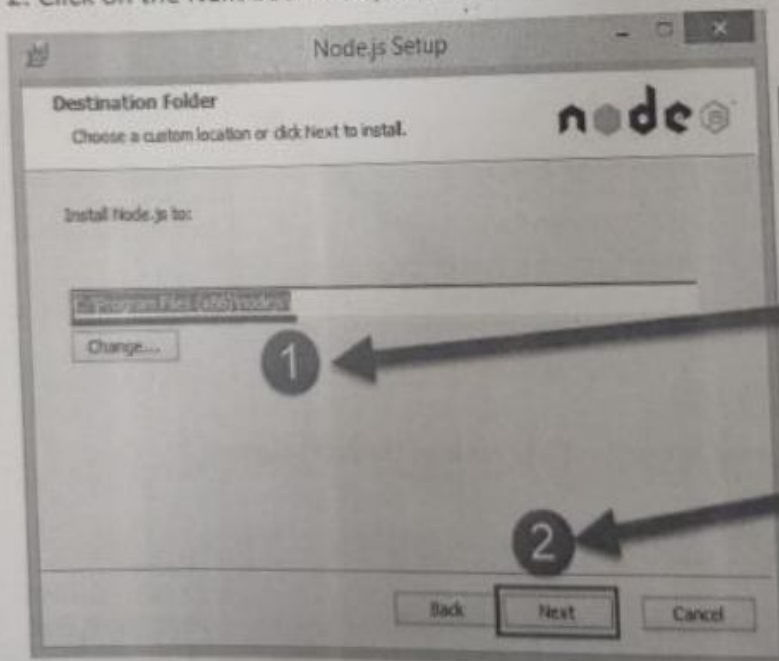


Accept the
Licence
Agreement
and Click
NEXT
Button

Step 5) Set up the path

In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

1. First, enter the file location for the installation of Node.js. This is where the files for Node.js will be stored after the installation.
2. Click on the Next button to proceed ahead with the installation.



Enter the
file
location for
installation

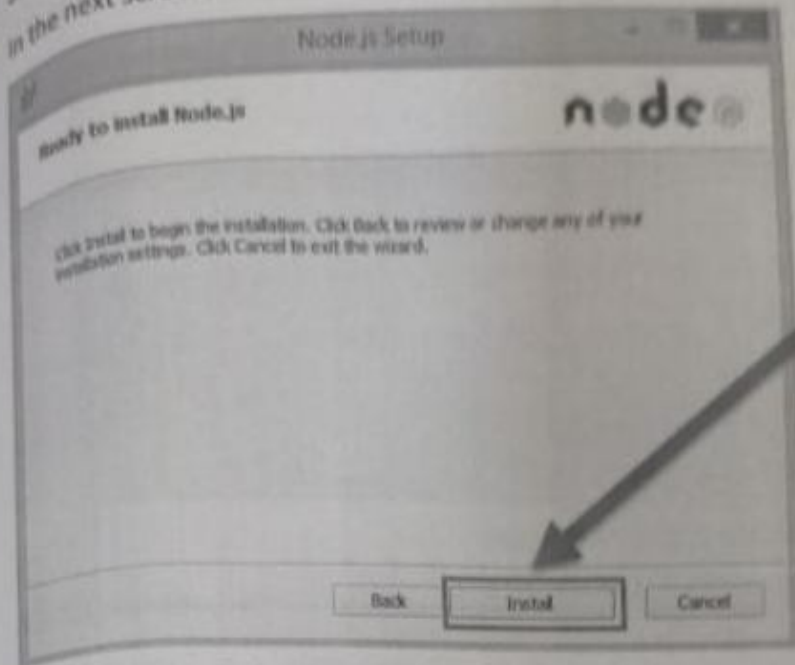
Click NEXT
Button

Step 6) Select the default components to be installed

Accept the default components and click on the Next button.

Step 7) Start the installation

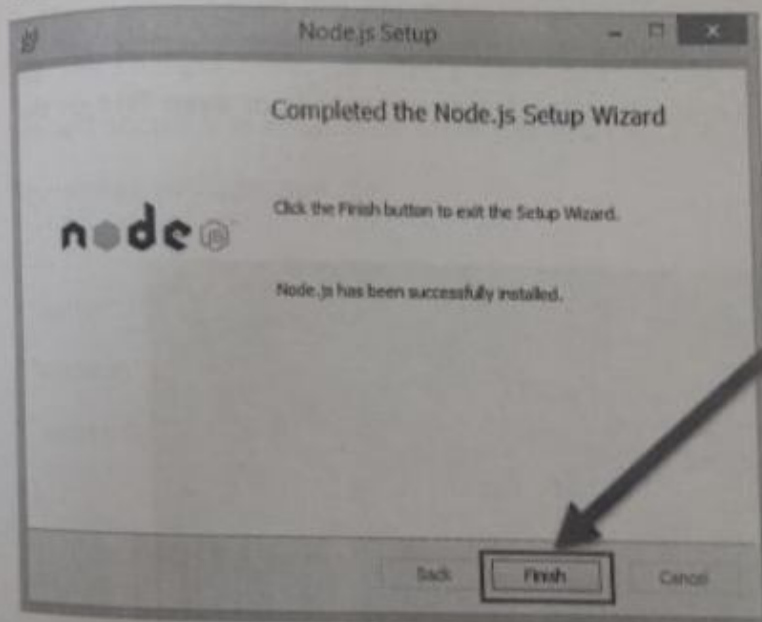
In the next screen, click the Install button to start installing Node.js on Windows.



Click the
INSTALL
Button to
Start
Installation

Step 8) Complete the installation

Click the Finish button to complete the installation.



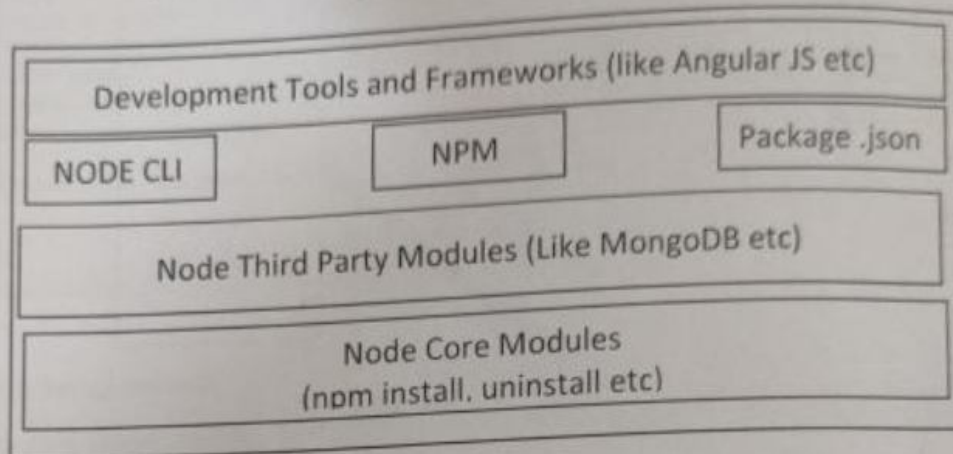
Click the
Finish
Button to
Complete
the
Installation

5.2.2 Components

- Node JS contains many components to develop, test and deploy applications.
- Following are the list of Node JS components:
 - Node CLI

- o NPM
- o package.json
- o Node Modules
- o Development Tools and Frameworks

The following figure depicts **NODE JS PLATFORM**:




5.2.2.1 Node CLI

Node JS has a CLI (Command Line Interface) to run the basic commands and script files.

This component gets installed by default while installing Node JS Platform.

We can open command prompt and type basic JavaScript commands or even files on the command prompt

 Node.js command prompt - node.exe

```
Your environment has been set up for using Node.js 16.13.0 (x64) and npm.
```

```
C:\Users\User>node.exe
```

```
Welcome to Node.js v16.13.0.
```

```
Type ".help" for more information.
```


```
> _
```

5.2.2.2 NPM

NPM stands for Node Package Manager. NPM is used to install, update, uninstall and configure Node JS modules/packages very easily.

When we install Node JS Base Platform, it installs only few components, modules and libraries like Node CLI, NPM etc. Later on, we can use NPM to upgrade Node JS with our required modules.

To check npm version, run "npm -v" command on Node CLI .:

 Node.js command prompt

```
C:\Users\User>npm -v  
8.1.0
```

```
C:\Users\User>
```

5.2.2.3 package.json

"package.json" is a plain text file in JSON format. It is used to manage our application required module dependencies. We should place this file in our application root folder.

It defines information like our application name, module dependencies, module versions etc. This configurations file is very important and requires more time to explain in detail.

We will discuss it in detail with some examples in coming posts.

Myownapp package.json file;

```
{  
  "name": "myownapp",  
  "version": "1.0",  
  "dependencies": {  
  }  
}
```

5.2.2.4 Node Modules

Node JS is a modular platform. Each functionality of it is implemented by a separate module or package. It has some core modules like npm, install, uninstall, update etc and rest all modules are third-party modules.

When we install Node JS Platform, by default only one module is installed i.e. npm module. We need to use "npm" command to install required modules one by one. All Core or Default modules are installed at /lib folder as *.js files. Node JS has thousands of modules. A full list of Node JS Platform's packages or modules can be found on the NPM website <https://npmjs.org/>.

5.2.2.5 Development Tools and Frameworks

Many companies have developed some tools and framework to ease and reduce the overhead of Node JS applications since the Node JS Platform has become very popular in developing Data-Sensitive Real-time and Network applications.

5.2.3 Request and response object

A request (an `http.IncomingMessage` object) and a response (an `http.ServerResponse` object) objects are the parameters of the callback function and are essential to handle the HTTP call.

Let us discuss each object in detail.

5.2.3.1 Request Object

The Request object is used to get information from a client and send to the server. It contains properties for the request query string, HTTP headers, parameters, body, etc. The properties of Request object, `req`, can be accessed through a dot(.) operator as `req.property`.

Few of its properties are listed as follows:

Property	description
<code>baseurl</code>	Denote the URL on which a router instance was mounted.
<code>body</code>	Hold key-value pairs of data submitted in the request body.
<code>cookies</code>	Used to read cookies data.
<code>hostname</code>	Specifies the hostname from the http header.
<code>path</code>	Specifies the path part of the request URL.

fresh	Specifies that the request is "fresh." It is the opposite of stale.
ip	Specifies the remote IP address of the request.
originalurl	Retains the original request URL, also allows to rewrite url for internal routing purposes.
params	Contain properties mapped to the named route parameters. For example, if you have the route as /user/:name, then the "name" property is available as req.params.name. This object defaults to {}.
protocol	The request protocol string, "http" or "https" when requested with TLS. (Transport Layer Security)
query	Contains a property for each query string parameter in the route.
route	The currently-matched route, a string.
secure	A Boolean that is true if a TLS connection is established.
stale	It indicates whether the request is "stale," and is the opposite of fresh property.
subdomains	It represents an array of subdomains in the domain name of the request.

5.2.3.2 Response Object

The Response object is used to send information from server to the client. I.e. the response object is going to client as response from the server. The methods of Response object, res, can be accessed through a dot(.) operator as res.method().

Following are few important methods of Request object.

Method name	Description
end()	Used to end the response process. It signals the server that the response is complete.
append()	Used to append data to the HTTP response header field.
attachment()	Used to send a file as an attachment in the HTTP response
cookie()	Used to set cookie name to value
format()	Used to content negotiation on the Accept HTTP header
get()	Used to provides HTTP response header specified by field.
set()	Set the response's HTTP header field .

<code>json()</code>	Returns the response in JSON format.
<code>jsonp()</code>	Used to returns the response in JSON format with JSONP support
<code>location()</code>	Based on the path parameter, it sets the response location HTTP header field.
<code>links()</code>	Used to set the response's Link HTTP header field by means of joining the links provided as properties of the parameter.
<code>redirect()</code>	Redirect request to given URL.
<code>send()</code>	Used to send the HTTP response.
<code>sendStatus()</code>	Used to set the response HTTP status code
<code>write()</code>	Writes text to client, i.e. write text in browser.
<code>writeHead()</code>	Sends status and response headers to the client
<code>type()</code>	Set the Content-Type HTTP header to the MIME type

5.3 Built-in modules

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files
- These modules can be reused throughout the Node.js application.
- Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.
- Node.js implements CommonJS modules standard.
- CommonJS is a group of helpers who define JavaScript standards for desktop, web server and console application.

Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Local Modules(user-defined modules)
3. Third Party Modules

5.3.1 Core Modules: Node.js has many built-in modules. They are part of the platform and comes along with Node.js installation. These modules can be loaded into the program by using the **require** function.

Syntax:

```
var module = require('module_name');
```

The *require()* function will return a JavaScript type depending on what the particular module returns. The following example demonstrates how to use the Node.js Http module to create a web server.

```
var http = require('http');
http.createServer(function(req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write("Advanced Web Technology!");
  res.end();}).listen(3000);
```

In the above example, the *require()* function returns an object because the Http module returns its functionality as an object. The function *http.createServer()* method will be executed when someone tries to access the computer on port 3000. The *res.writeHead()* method is the status code where 200 means it is OK, while the second argument is an object containing the response headers.

The following list contains some of the important core modules in Node.js:

Core Modules	Description
http	creates an HTTP server in Node.js.
fs	used to handle file system.
path	includes methods to deal with file paths.
querystring	used for parsing and formatting URL query strings.

os	provides information about the operating system.
process	provides information and control about the current Node.js
url	provides utilities for URL resolution and parsing.
assert	set of assertion functions useful for testing.

5.3.2 Local Modules(User Defined Modules):

Local modules are created locally in your Node.js application. You can create your own modules, and easily include them in your applications.

The following example demonstrates how to create a user-defined module and include it in our node.

Example:

Create a calc.js file that has the following code:

```
exports.add= function(x, y) {
    return x + y;
};
exports.sub = function(x, y) {
    return x - y;
};
exports.mult = function(x, y) {
    return x * y;
};
exports.div = function(x, y) {
    return x / y;
};
```

The `exports` keyword is used to make properties and methods available outside the module file. Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the `require()` function.

Now you can include and use the module in any of your Node.js files. Let's create a file with name `implement_calc.js`

Filename: `implement_calc.js`

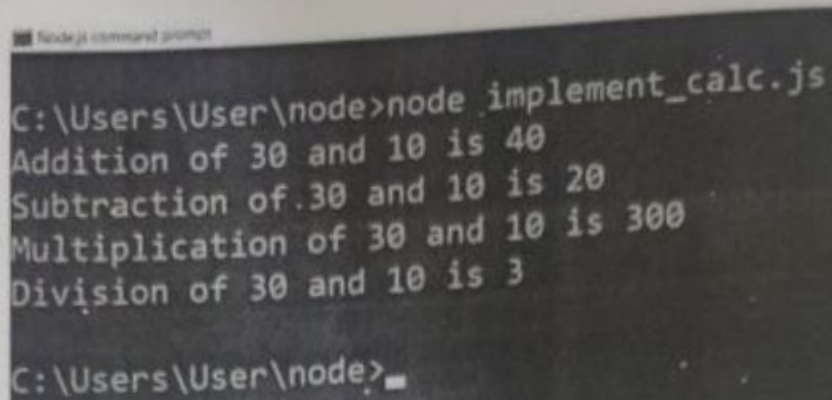
```
var calculator = require('./calc');  
var x = 30, y = 10;  
console.log("Addition of 30 and 10 is "  
    + calculator.add(x, y));  
console.log("Subtraction of 30 and 10 is "  
    + calculator.sub(x, y));  
console.log("Multiplication of 30 and 10 is "  
    + calculator.mult(x, y));  
console.log("Division of 30 and 10 is "  
    + calculator.div(x, y));
```

The use `./` to locate the module, that means that the module is located in the same folder as the `implement_calc.js` file.

Save the code above in a file called "`implement_calc.js`", and initiate the file:

```
node implement_calc.js
```

Output:



```

C:\Users\User\node>node implement_calc.js
Addition of 30 and 10 is 40
Subtraction of 30 and 10 is 20
Multiplication of 30 and 10 is 300
Division of 30 and 10 is 3
C:\Users\User\node>

```

Note: This module also hides functionality that is not needed outside of the module.

5.3.3 Third-party modules: Third-party modules are modules that are available online using the Node Package Manager(NPM). These modules can be installed in the project folder or globally. Some of the popular third-party modules are mongoose, express, angular, and react.

Example:

- npm install express
- npm install mongoose
- npm install -g @angular/cli

5.3.4 HTTP Module

- Node.js has a built-in module called HTTP, which allows Node.js to transfer data over the Hyper Text Transfer Protocol (HTTP).
- To include the HTTP module, use the *require()* method:

```
var http = require('http');
```
- The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.
- The *createServer()* method is used to create an HTTP server:

Example:

```

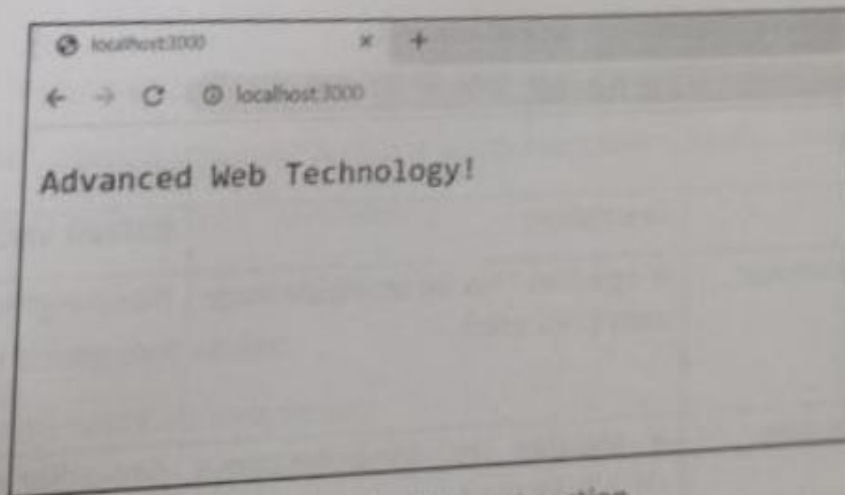
var http = require('http');
//create a server object:

```

```
http.createServer(function (req, res) {  
  res.write('Advanced Web Technology!'); //write a response to the client  
  res.end(); //end the response  
}).listen(3000); //the server object listens on port 3000
```

- The function passed into the `http.createServer()` method, will be executed when someone tries to access the computer on port 3000.
- Save the code above in a file called "http_demo.js", and initiate the file:
- Run demo_http.js: using `node demo_http.js` on command prompt.
- If you type <http://localhost:3000> in URL bar of any browser, you will get following output:

Output:



Note: This module will be discussed in detail in next section

5.4 Node.js as a Web Server

Web server is required to access the web pages of any web application. The web server handles all the http requests coming from the web application. For example, Apache web server is used for PHP web applications, and IIS web server is used for ASP.NET web applications.

Node.js allows to create own web server that handle incoming HTTP requests asynchronously. Node.js has a built-in module called HTTP. This module enables Node.js to send data over HTTP. The HTTP module can create an HTTP server that listens to server ports and gives a response back to the client.

Methods:

method	description
createServer()	create HTTP server
writeHead()	Add HTTP header to the request

5.4.1 Creating Web Server and Adding HTTP header:**5.4.1.1 createServer() method:**

createServer() is used to create HTTP server. It returns an HTTP Server object. This method includes a request object, which is used to obtain information about the current HTTP request, such as request header, the url, and data.

Syntax

```
http.createServer([options][, requestListener]);
```

Where, parameters are as follows:

(i) options:

options	description	Default value
IncomingMessage	It specifies the IncomingMessage class to be used.	IncomingMessage
ServerResponse	It specifies the ServerResponse class to be used.	ServerResponse
insecureHTTPParser	When it is set to true, It use an insecure HTTP parser that accepts invalid HTTP headers.	false
maxHeaderSize	It specifies the maximum length of request headers in bytes.	16384 (16 KB).

(ii) requestListener: This parameter is optional. It specifies a function that will be called whenever the server receives a request. This function is known as a *requestListener*. requestListener handles user requests and gives response back to the user.

5.4.1.2 writeHead() method:

writeHead() method write a header to the response. It is also used to specify the status code and the content type.

Syntax:

```
writeHead(status_code[, status_message][, headers]);
```

where

status_code: It is a 3-digit HTTP status code. For example, 404 for file not found, 200 for ok, 505 for Internal Server Error.

status_message: It is string that represents the status message

headers: It takes any function, string or array.

Return value: It returns a reference to the ServerResponse, so that calls can be chained

For example: Following code set content header. Here, first argument is status code 200, means all is "ok". Second argument is object which represent type of contain.

```
writeHead(200, {'Content-Type': 'text/html'});
```

5.4.1.3 How to create web server

Use createServer() to create web server.

The following example demonstrates how to use Node.js http module to create a web server. This code creates a server that will listen on port number 8000. If a browser request is made on this port number, the server will respond to the client. The function passed in the createServer() will be executed when the client visit the url http://localhost:8000

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

```
// Import http module
```

```
// create a server object:
```

```
var server = http.createServer(function(req, res) {
```

```
// create a server object and call  
function with request and  
response object
```

```
{
```

```
// code
```

```
}
```

```
// write code here
```

```
);  
server.listen(8000); //Listen to port number 8000
```

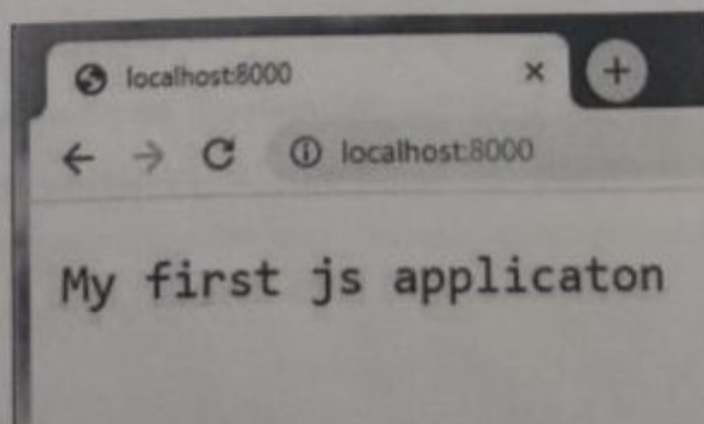
Example: The following code create web server and display message "My first js application" in browser.

```
var http = require('http'); //import http module  
    //create a server object:  
http.createServer(function (req, res)  
{  
    res.write('My first js application'); //write a response to the client  
    res.end(); //ending the response  
}).listen (8000); // server is set to listen on the 8000 port
```

Steps to run the code:

1. Save the code in a file with .js extension, say *test.js*.
2. Open the command prompt
3. Using the *cd* command, navigate to the folder containing the file.
4. Run the command: *node file_name.js* . Here, write *nodetest.js*
5. Open the browser
6. Type URL *http://localhost:8000* in URL bar.

Output: In browser, output will be look like:



Above example can be written as

```
var http = require('http');  
var server = http.createServer(function (req, res)  
{  
    res.write('My first jsapplicaton');  
    res.end();  
});  
server.listen(8000);
```

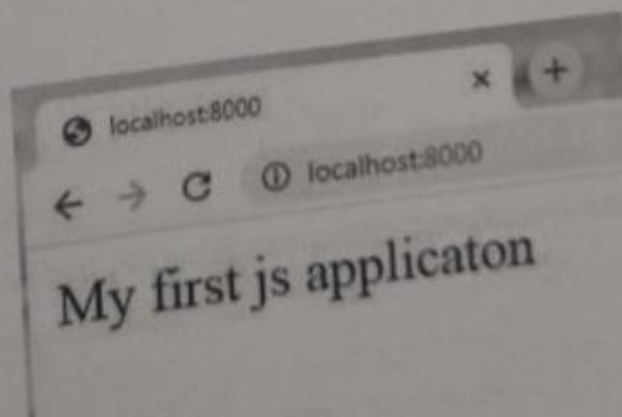
5.4.1.4 How to add a HTTP Header

Use writeHead() method to add header.

Example: Following code add http header.

```
var http = require('http');  
//create a server object:  
http.createServer(function (req, res)  
{  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write('My first jsapplicaton'); //write a response to the client  
    res.end(); //end the response  
}).listen(8000); //the server object listens on port 8080
```

Output :



5.4.2 Query String

Query strings are appended to the end of a URL followed by ?. The query string contains parameters in the form of a key-value pair. The key-value pair is separated by the equal (=) symbol. The ampersand (&) symbol separates each parameter.

For example,

- (1) `https://jump2learn.com?book=advanced` // here book is key and advanced is value
- (2) `https://jump2learn.com?book=advanced&price=200` // it contains two parameters

5.4.2.1 Reading Query string data

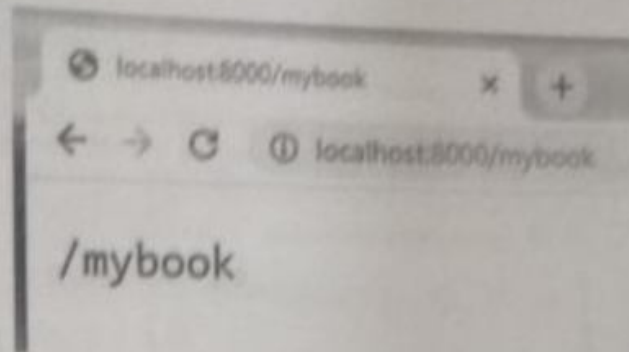
The function, which is passed into the `createServer()`, has first argument as request object. This request object represents the request from the client. The Request object is used to get information from a client and send information to server. The request object has properties for the request query string, body, parameters, body, HTTP headers etc. "url" property of request object contains the portion of the url that follows the domain name. For example, if user type `https://jump2learn.com/userguide` in browser, then the value of `url` property is `"/userguide"`

Example: consider following code.

```
var http = require('http');
http.createServer(function (req, res)
{
    res.write(req.url);
    res.end();
}).listen(8000)
```

Output:

If user type `http://localhost:8000/mybook` in the URL bar, browser will display:



5.4.2.2 Splitting Query string

Splitting of query string data means parse or separate query string parameter, so they can process individually. To split query string data we need to first import "url" module, use `url.parse` method and query property.

`url.parse()`:

The `url.parse()` method parses a URL string. It split various elements of the URL such as host, pathname, search keys, etc.

Syntax:

`url.parse(urlString[, parseQueryString])`

where,

`urlString`: The URL string to parse.

`parseQueryString`: It is Boolean value. It indicates whether the method should parse the query string. If it is true, it parses query string. If it is false, it will return URL object unparsed, undecoded string. Its Default value is false.

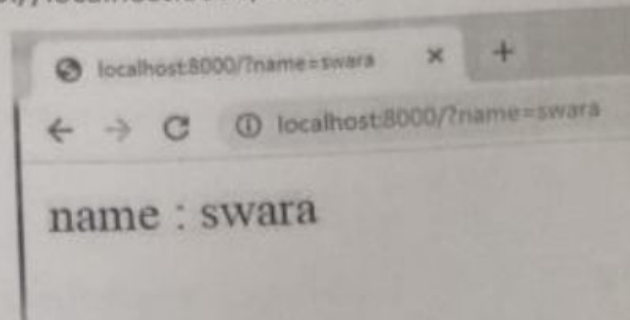
`query`:

`.query` property: It return object containing a property for each query string parameter.

Example: Following code read query string data "name".

```
var http = require('http');
var url = require('url');           //import url module
http.createServer(function (req, res)
{
  res.writeHead(200, {'Content-Type': 'text/html'});
  var query_data = url.parse(req.url, true).query;
  res.write("name : " + query_data.name);
}).listen(8000);
```

Output: If you type `http://localhost:8000/?name=swara` in URL bar, it will display:



Explanation:

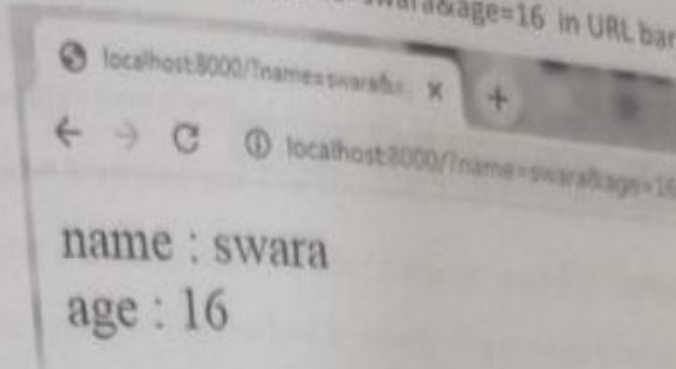
- Script line `var url = require('url')` import `url` module and return reference to "`url`" variable.
- Script line `var query_data = url.parse(req.url, true).query;` is important. Here,
 - `req.url` will hold `/?name=swara`.
 - `url.parse` split the various elements of the `req.url`.
 - `.query` property return query string data.
- `query_data.name` refer to "name" key in query string.

Reading multiple data

Following code parse multiple query string data.

```
var http = require('http');
var url = require('url');
http.createServer(function (req, res)
{
    res.writeHead(200, {'Content-Type': 'text/html'});
    var query_data = url.parse(req.url, true).query;
    res.write("name : " + query_data.name);
    res.write("<br/>");
    res.write("age : " + query_data.age);
    res.end();
}).listen(8000);
```


Output: If you type `http://localhost:8000/?name=swara&age=16` in URL bar, it will display:



5.5 File System modules

- The Node.js file system module enables us to work with the file system on one's own computer.
- To include the File System module, we use the `require()` method as below:
- `var fs = require('fs');`
- Every method in the 'fs' module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error.
- It is better to use an asynchronous method instead of a synchronous method,
- As the asynchronous method never blocks a program during its execution, whereas the second one does.
- Some common uses of 'fs' module are:
 - Read Files
 - Create Files
 - Update Files
 - Delete Files
 - Rename Files
- To demonstrate the above uses let us first create a simple text file and save it as `test.txt`. Consider that the `test.txt` has following content:

This is a test file for demonstrating the use of different uses of file system modules of node js in the book Advanced Web Technology!!

5.5.1 Read Files

- The `fs.readFile()` method is used to read files on one's own computer.
- Let us create a js file named `read_demo.js` with the following code:

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

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

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

- Run the above code as follows:

```
node read_demo.js
```

Output:

```
C:\Users\User\node>node read_demo.js
Synchronous read: This is a test file for demonstrating the use of
different uses of file system modules of node js
in the book Advanced Web Technology!!
Program Completed
Asynchronous read: This is a test file for demonstrating the use of
different uses of file system modules of node js
in the book Advanced Web Technology!!
```

5.5.2 Create Files

- The File System module of Node js has following methods for creating new files:

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

5.5.2.1 Opening a file

- fs.open() method is used to open a file for different purpose depending on the flag.
- Following is the syntax of the method to open a file in asynchronous mode-
`fs.open(path, flags[, mode], callback)`
- In the above syntax :
 - path – filename including path that needs to be opened.
 - flags – behavior of the file to be opened. All possible values have been listed below.
 - mode – It sets the file mode (permission), but only when the file was created. It defaults to 0666, readable and writeable.
 - callback – This is the callback function which gets two arguments (err, fd). err- error while performing any operation on file and fd is file data
- Flags for read/write operations are -

Flag.	Description (Open file for)
r	Reading.
r+	Reading and writing.
rs	Reading in synchronous mode.
rs+	Reading and writing, asking the OS to open it synchronously.
w	Writing. The file is created (if it does not exist) or truncated (if it exists).
wx	Like 'w' but fails if the path exists.
w+	Reading and writing. The file is created (if it does not exist) or truncated
wx+	Like 'w+' but fails if path exists.
a	Appending. The file is created if it does not exist.
ax	Like 'a' but fails if the path exists.

a+	Reading and appending. The file is created if it does not exist.
ax+	Like 'a+' but fails if the path exists.

Example

The following code shows how to open a file test.txt for reading and writing. Save this file as open_demo.js.

```
var fs = require("fs");
// Asynchronous - Opening File
console.log("Trying to open file!");
fs.open('test.txt', 'r+', function(err, fd) {
  if (err) {
    return console.error(err);
  }
  console.log("File opened successfully!");
});
```

Run the open_demo.js to get the following output:

```
$ node open_demo.js
```

Output:

```
C:\Users\User\node>node open_demo.js
Trying to open file!
File opened successfully!
```

5.5.2.2 Writing in a file:

- `fs.writeFile()` method is used to create a new file if it does not exist. It will over-write the file if the file already exists.
- Syntax:

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

- path – string having the filename including path.

- data – The string or buffer to be written into the file.
- options – It's 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 that returns an error in case of any writing error.

Example

The following code shows how to create and write in new file named test2.txt for reading and writing. Save this file as write_demo.js.

```
var fs = require("fs");
console.log("Trying to write in a new file");
fs.writeFile('test2.txt', 'Easy Learning! With Advanced Web Technology! book',
function(err) {
    if (err) {
        return console.error(err);
    }
    console.log("Data written successfully!");
    console.log("Let's read newly written data");
    fs.readFile('test2.txt', function (err, data) {
        if (err) {
            return console.error(err);
        }
        console.log("Asynchronous read: " + data.toString());
    });
});
```

Output:

```
C:\Users\User\node>node write_demo.js
Trying to write in a new file
Data written successfully!
Let's read newly written data
Asynchronous read: Easy Learning! With Advanced Web Technology! book
```

5.5.2.3 Appending in a file

- *fs.appendFile()* method is also used to create a new file if it does not exist. It will append the content in the file if it already exists.
- **Syntax:**
 - `fs.appendFile(filename, data[, options], callback)`
- **path** – string having the filename including path.
- **data** – The string or buffer to be written into the file.
- **options** – It's an object which will hold {encoding, mode, flag}. By default, encoding is utf8, mode is octal value 0666, and flag is 'a'.
- **callback** – This is the callback function which gets a single parameter *err* that returns an error in case of any writing error.
- **Example**
- The following code shows how to create and write in new file named test3.txt for reading and creating file using `appendFile()` method of `fs`. Save this file as `append2create_demo.js`.

```

    ▪ var fs = require("fs");

console.log("Trying to create a new file with appendFile() method of File System");
fs.appendFile('test3.txt', 'Advanced Web Technology book by Jump2Learn Publishing! ',
function(err) {
    if (err) {
        return console.error(err);
    }
    console.log("Data written using append method successfully!");
    console.log("Let's read newly written data");
    fs.readFile('test3.txt', function (err, data) {
        if (err) {
            return console.error(err);
        }
        console.log("Asynchronous read: " + data.toString());
    });
}

```



```
});
```

```
});
```

Output:

```
C:\Users\User\node>node append2create_demo.js
Trying to create a new file with appendFile() method of File System
Data written using append method successfully!
Let's read newly written data
Asynchronous read: Advanced Web Technology book by Jump2Learn Publishing!
```

5.5.3 Update Files

- The File System module has methods for updating files:
 - fs.appendFile()
 - fs.writeFile()
- The fs.appendFile() method appends the specified content at the end of the specified file:
- Example
- The following code append the text "Very Informative and Good Book." to the end of the file "text3.txt". Save this file as append2update_demo.js.

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

console.log("Trying to update an existing file with appendFile() method of File
System");

fs.appendFile('test3.txt', 'Very Informative and Good Book!', function(err) {

    if (err) {

        return console.error(err);

    }

    console.log("Data appended using append method successfully!");
    console.log("Let's read newly updated data");
    fs.readFile('test3.txt', function (err, data) {

        if (err) {

            return console.error(err);

        }

    })
})
```

```

    console.log("Asynchronous read: " + data.toString());
  });
});

```

Output:

```

C:\Users\liber\node>node appendUpdate_demo.js
Trying to update an existing file with appendFile() method of File System
Data appended using append method successfully!
Let's read newly updated data
Asynchronous read: Advanced Web Technology book by Jump2Learn Publishing! Very Informative and Good Book!

```

- The fs.writeFile() method replaces the specified file and its content.
- **Example:**
- The following code overwrites the content of the file "text3.txt". Save this file as write2update_demo.js.

```

var fs = require("fs");

console.log("Trying to overwrite an existing file with writeFile() method of File
System");

fs.writeFile('test3.txt', 'New Overwritten text! ', function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("Data overwrite using write method successfully!");
  console.log("Let's read newly overwritten data of same file");
  fs.readFile('test3.txt', function (err, data) {
    if (err) {
      return console.error(err);
    }
    console.log("Asynchronous read: " + data.toString());
  });
});

```

Output:

101

```
C:\Users\User\node>node write2update_demo.js
Trying to overwrite an existing file with writeFile() method of File System
Data overwrite using write method successfully!
Let's read newly overwritten data of same file
Asynchronous read: New Overwritten text!
```

5.5.4 Delete Files

- The File System module has `unlink()` method for deleting files.

- **Syntax:**

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

- path – string having the filename including path
- callback – This is the callback function which gets a single parameter `err` that returns an error in case of any deleting error

- **Example:**

- The following code deletes the file "text3.txt". Save this file as `delete_demo.js`.

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

console.log("Trying to delete an existing file");
fs.unlink('test3.txt', function(err) {
  if (err) {
    return console.error(err);
  }
  console.log("File deleted successfully!");
});
```

Output:

```
C:\Users\User\node>node delete_demo.js
Trying to delete an existing file
File deleted successfully!
```


5.5.5 Rename Files

- The `fs.rename()` method is used to asynchronously rename a file at the given old path to a given new path. It will overwrite the destination file if it already exists.

- **Syntax:**

```
fs.rename( oldPath, newPath, callback )
```

- `oldpath` – path of the file that has to be renamed. It can be a string, buffer or URL
- `newPath` - new path that the file has to be renamed. It can be a string, buffer or URL.
- `callback` – This is the callback function which gets a single parameter `err` that returns an error in case of any renaming

- **Example:**

- The following code renames the file "text3.txt" to "temp.txt". Save this file as `rename_demo.js`.

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

console.log("Trying to rename an existing file");
fs.rename('test3.txt', 'temp.txt', function (err) {
  if (err) {
    return console.error(err);
  }
  console.log("File renamed successfully!");
});
```

Output:

```
C:\Users\User\node>node rename_demo.js
Trying to rename an existing file
File renamed successfully!
```