

## CHAPTER NO 3

### NODE JS

#### 1. Introduction

- Node.js is an open source, cross-platform runtime environment and library that is used for running web applications outside the client's browser.
- Node.js runs the V8 JavaScript engine, the core of Google Chrome, outside of the browser.
- Ryan Dahl developed it in 2009, and its latest iteration, version 15.14, was released in April 2021.
- Developers use Node.js to create server-side web applications, and it is perfect for data-intensive applications since it uses an asynchronous, event-driven model.
- Node.js is used by big business and small enterprises alike. While the likes of Amazon, Netflix, eBay, Reddit and Paypal all use it, 43% of developers using Node.JS in 2023 do so for enterprise applications.

#### 2. Features of Node JS

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

#### 3. Installation of node js

- You can download the latest version of Node.js installable archive file from <https://nodejs.org/en/>
- **check version:** node -v
- To execute javascript code outside of browser use **node** command
- C:\AIT>node (enter)
- This command starts node environment called REPL

#### 4. Node.js First Example

**demo.js**

```
var a=10;  
console.log(a)
```

**Open command prompt and run the following code:**

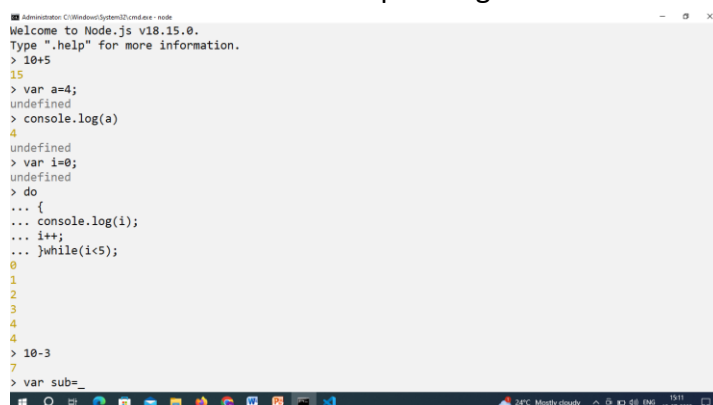
```
d:\AIT\nodejs_ex> node demo.js
```

**Output:**

10

#### 5. REPL

- REPL (READ, EVAL, PRINT, LOOP) is a computer environment similar to Shell (Unix/Linux) and command prompt.
- Node comes with the REPL environment when it is installed.
- System interacts with the user through outputs of commands/expressions used.
- It is useful in writing and debugging the codes.
- The work of REPL can be understood from its full form:
  - **Read** : It reads the inputs from users and parses it into JavaScript data structure. It is then stored to memory.
  - **Eval** : The parsed JavaScript data structure is evaluated for the results.
  - **Print** : The result is printed after the evaluation.
  - **Loop** : Loops the input command. To come out of NODE REPL, press ctrl+c twice
- **Getting Started with REPL:**
  - To start working with REPL environment of NODE; open up the terminal (in case of UNIX/LINUX) or the Command prompt (in case of Windows) and write node and press 'enter' to start the REPL.
  - The REPL has started and is demarcated by the '>' symbol. Various operations can be performed on the REPL.
  - Below are some of the examples to get familiar with the REPL environment.



```
Administrator: C:\Windows\System32\cmd.exe - node  
Welcome to Node.js v18.15.0.  
Type ".help" for more information.  
>  
> 10+5  
15  
> var a=4;  
undefined  
> console.log(a)  
4  
undefined  
> var i=0;  
undefined  
> do  
... {  
... console.log(i);  
... i++;  
... }while(i<5);  
0  
1  
2  
3  
4  
5  
> 10-3  
7  
> var sub=_  
undefined
```

## 6. NPM (Node Package Manager)

- It is the default package manager for Node.js and is written entirely in Javascript.
- Developed by Isaac Z. Schlueter, it was initially released in January 12, 2010.
- NPM manages all the packages and modules for Node.js and consists of command line client npm.
- It gets installed into the system with installation of Node.js.
- The required packages and modules in Node project are installed using NPM.
- A package contains all the files needed for a module and modules are the JavaScript libraries that can be included in Node project according to the requirement of the project.
  
- **Installing NPM:**
- To install NPM, it is required to install Node.js as NPM gets installed with Node.js automatically.
- **Checking and updating npm version:**
- `npm -v`
  
- **Installing Modules using npm**
- `npm install <Module Name>`
  
- **Global vs Local Installation**
- By default, npm installs dependency in local mode. Here local mode specifies the folder where Node application is present.
- In the global mode, NPM performs operations which affect all the Node.js applications on the computer whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.
  
- **1. Install Package Locally**
- Use the following command to install any third party module in your local Node.js project folder.
- `C:\>npm install <package name>`
- For example, the following command will install ExpressJS into MyNodeProj folder.
- `C:\MyNodeProj> npm install express`
- All the modules installed using NPM are installed under **node\_modules** folder.
  
- **2. Install Package Globally**
- NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages.
- NPM installs global packages into `/<User>/local/lib/node_modules` folder.

- Apply **-g** in the install command to install package globally. For example, the following command will install ExpressJS globally.
- C:\MyNodeProj> npm install -g express

- **3. Uninstall Packages**

- Use the following command to remove a local package from your project.
- C:\>npm uninstall <package name>
- The following command will uninstall ExpressJS from the application.
- C:\MyNodeProj> npm uninstall express

## 7. How modules work

- Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which 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 Module Types**
- Node.js includes three types of modules:
  - Core Modules
  - Local Modules
  - Third Party Modules
- **1. Core Modules:**
- Node.js has many built-in modules that are part of the platform and come with Node.js installation. These modules can be loaded into the program by using the required function.
- **Syntax:**
- `const 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.
- **Example:**
- `const http = require('http');`
- `http.createServer(function (req, res) {`
- `res.writeHead(200, { 'Content-Type': 'text/html' });`
- `res.write('Welcome to this page!');`
- `res.end();`
- `}).listen(3000);`
- The following list contains some of the important core modules in Node.js:

Modules	Description
http	creates an HTTP server in Node.js.
assert	set of assertion functions useful for testing.
fs	used to handle file system.
path	includes methods to deal with file paths.
process	provides information and control about the current Node.js process.
os	provides information about the operating system.
querystring	utility used for parsing and formatting URL query strings.
url	module provides utilities for URL resolution and parsing.

- **2. Local Modules:**

- Local Modules: Unlike built-in and external modules, local modules are created locally in your Node.js application. Let's create a simple calculating module that calculates various operations. Create a calc.js file that has the following code:

- **Filename: calc.js**

- 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;
- };
- Since this file provides attributes to the outer world via exports, another file can use its exported functionality using the require() function.

- **Filename: index.js**
- `const calculator = require('./calc');`
- `let x = 50, y = 10;`
- `console.log("Addition of 50 and 10 is " + calculator.add(x, y));`
- `console.log("Subtraction of 50 and 10 is " + calculator.sub(x, y));`
- `console.log("Multiplication of 50 and 10 is " + calculator.mult(x, y));`
- `console.log("Division of 50 and 10 is " + calculator.div(x, y));`
- Step to run this program: Run the index.js file using the following command:
- `node index.js`
- **Output:**
- Addition of 50 and 10 is 60
- Subtraction of 50 and 10 is 40
- Multiplication of 50 and 10 is 500
- Division of 50 and 10 is 5
- **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`

## 8. Webserver Creation

When you view a webpage in your browser, you are making a request to another computer on the internet, which then provides you the webpage as a response.

That computer you are talking to via the internet is a web server.

A web server receives HTTP requests from a client, like your browser, and provides an HTTP response, like an HTML page or JSON from an API.

The Node.js framework is mostly used to create server-based applications.

There are a variety of modules such as the “http” and “request” module, which helps in processing server related requests in the webserver space.

## Creating a Web Server using Node:

When you installed Node.js, the core library was installed too. Within that library is a module called http.

Example:



run your web server using node app.js. Visit <http://localhost:7000> and you will see a message saying "Hello World".

### Steps:

1. start by loading the http module that's standard with all Node.js installations. Add the following line to hello.js:  
`const http = require("http");`
2. The http module contains the function to create the server. The next step is to call `createServer()` method of http and specify callback function with request and response parameter.

A callback is an anonymous function (a function without a name) that's set up to be invoked as soon as another function completes.

The `http.createServer()` method includes request and response parameters which is supplied by Node.js. The request object can be used to get information about the current HTTP request e.g., url, request header, and data. The response object can be used to send a response for a current HTTP request.

To send a response, first it sets the response header using `writeHead()` method and then writes a string as a response body using `write()` method.

```
var server = http.createServer(function (request, response) {  
  // creating server handle incoming requests here..  
});
```

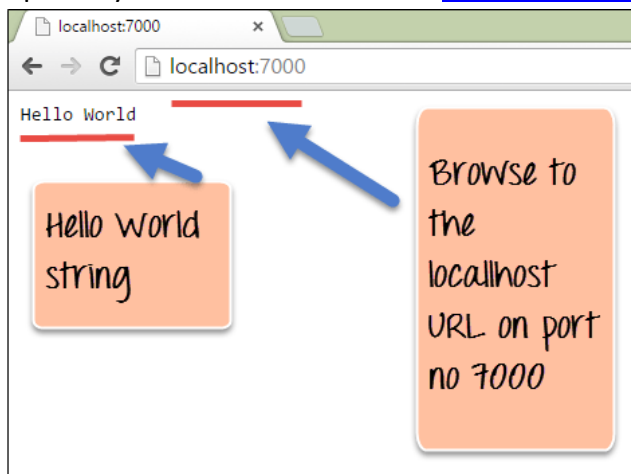
3. `response.writeHead(200);` sets the HTTP status code of the response. HTTP status codes indicate how well an HTTP request was handled by the server. In this case, the status code 200 corresponds to "OK".
4. `response.end("Hello World!");`, writes the HTTP response back to the client who requested it. This function returns any data the server has to return. In this case, it's returning text data.
5. After we create our server, we must bind it to a network address. We do that with the `server.listen()` method. It accepts three arguments: port, host, and a callback function that fires when the server begins to listen.

All of these arguments are optional, but it is a good idea to explicitly state which port and host we want a web server to use. When deploying web servers to different environments.

```
server.listen((7000), () => {  
  console.log("Server is Running");
```

6. Running the application:

open any browser to the address <http://localhost:7000>



Example 2: server.js

```
var http = require('http'); // Import Node.js core module
```

```
var server = http.createServer(function (req, res) { //create web server  
  if (req.url == '/') { //check the URL of the current request
```

```
    // set response header  
    res.writeHead(200, { 'Content-Type': 'text/html' });
```

```
    // set response content
```



```

    res.write('<html><body><p>This is home Page.</p></body></html>');
    res.end();

}
else if (req.url == "/student") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is student Page.</p></body></html>');
    res.end();

}
else if (req.url == "/admin") {

    res.writeHead(200, { 'Content-Type': 'text/html' });
    res.write('<html><body><p>This is admin Page.</p></body></html>');
    res.end();

}
else
    res.end('Invalid Request!');

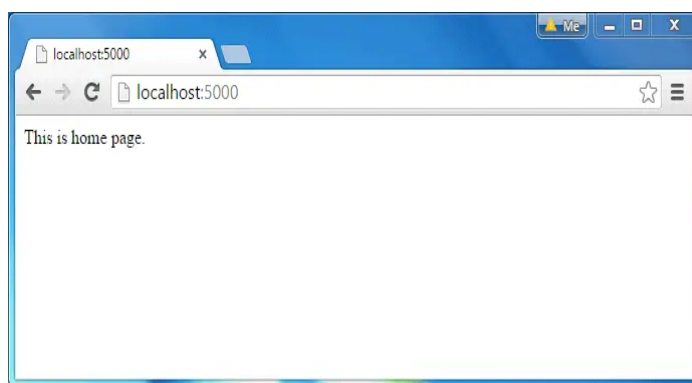
});

server.listen(5000); //6 - listen for any incoming requests

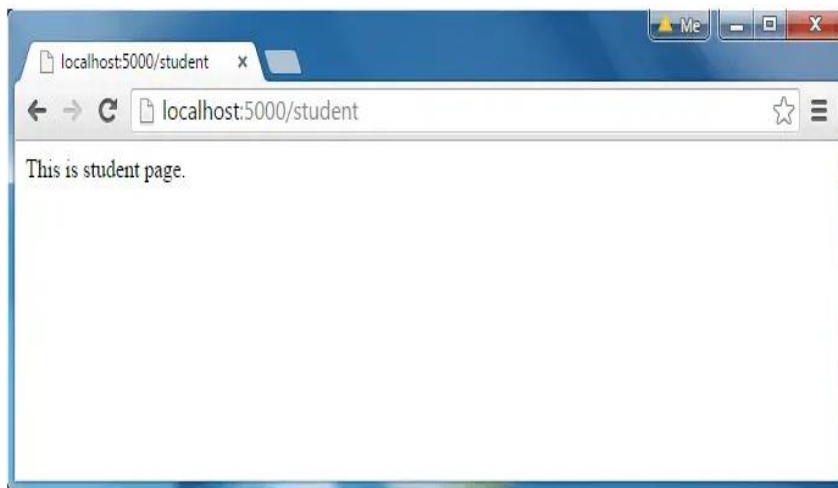
console.log('Node.js web server at port 5000 is running..')

```

Output:



Node.js Web Server Response



Node.js Web Server Response

## 9. Events in Node JS

Node.js has an event-driven architecture which can perform asynchronous tasks.

Node.js has 'events' module which emits named events that can cause corresponding functions or callbacks to be called.

Functions(Callbacks) listen or subscribe to a particular event to occur and when that event triggers, all the callbacks subscribed to that event are fired one by one in order to which they were registered.

**The EventEmitter class:** All objects that emit events are instances of the EventEmitter class. The event can be emitted or listen to an event with the help of EventEmitter.

### Syntax:

```
const EventEmitter=require('events');
```

```
var eventEmitter=new EventEmitter();
```

### Listening events:

Before emits any event, it must register functions(callbacks) to listen to the events.

### Syntax:

```
eventEmitter.addListener(event, listener)
```

```
eventEmitter.on(event, listener)
```

```
eventEmitter.once(event, listener)
```

eventEmitter.on(event, listener) and eventEmitter.addListener(event, listener) are pretty much similar.

It adds the listener at the end of the listener's array for the specified event.

Multiple calls to the same event and listener will add the listener multiple times and correspondingly fire multiple times.

Both functions return emitter, so calls can be chained.

eventEmitter.once(event, listener) fires at most once for a particular event and will be removed from listeners array after it has listened once.

Returns emitter, so calls can be chained.

**Emitting events:** Every event is named event in nodejs. We can trigger an event by emit(event, [arg1], [arg2], [...]) function. We can pass an arbitrary set of arguments to the listener functions.

#### **Syntax:**

```
eventEmitter.emit(event, [arg1], [arg2], [...])
```

#### **Example:**

##### **// Importing events**

```
const EventEmitter = require('events');
```

##### **// Initializing event emitter instances**

```
var eventEmitter = new EventEmitter();
```

##### **// Registering to myEvent**

```
eventEmitter.on('myEvent', (msg) => {  
  console.log(msg);  
});
```

##### **// Triggering myEvent**

```
eventEmitter.emit('myEvent', "First event");
```

#### **Methods in Events**

**1. emit(event, [arg1], [arg2], [arg3],....[]):** emit() is used to raise an event. The first argument event is the name of the event followed by any number of parameters.

Example: customEmitter.emit("newLine", "A custom newLine event has been emitted",true);

The emit method returns true if one or more listeners are attached. If there are no listeners attached for a particular event, then the emit method returns false.

**2. on(event, listener):** on() listens to the event and executes the event handler. The listener is the event handler.

**Syntax:** emitter.on(eventName,listener)

The eventName is the string identifying the named event while the listener is the callback function.

**Example:**

```
const EventEmitter = require('events');
const customEmitter = new EventEmitter();
customEmitter.on("newLine", (arg1,arg2)=>{
    console.log(arg1);
    console.log(arg2);
});
customEmitter.emit("newLine", "A custom newLine event has been emitted",true);
```

**3. once(event, listener):** to handle events just one time.

Syntax: emitter.once(eventName, listener)

**Example:**

```
const EventEmitter = require('events');
const customEmitter = new EventEmitter();
customEmitter.once('event1', (data) => console.log(data));
customEmitter.emit('event1', 'event 1 emitted');
customEmitter.emit('event1', 'event 1 emitted');
```

**4. addListener(event, listener):** addListener() adds a listener to the end of listeners array for the specified event.

Syntax: eventEmitter.addListener('test-event', ()=> { console.log ("test one") } );

**5. removeListener(event, listener):** removeListener() removes a listener from the listener's array for the specified event.

Syntax: eventEmitter.removeListener('myEvent', fun1);

**6. removeAllListeners([event]):** removeAllListeners() removes all the listeners of the specified events.

Syntax: eventEmitter.removeAllListeners('myEvent');

**7. listeners(event):** listeners() returns an array of the specified event.

Syntax: eventEmitter.listeners(event)

**8. setMaxListeners(n):** If more than 10 listeners are added to the event, the EventEmitter class gives a warning by default. This function allows the number of listeners to increase, in order to do so set the value to zero for unlimited.

Syntax: eventEmitter.setMaxListeners(n);

**Steps:**

**Step 1:** Import the “events”.

**Step 2:** Create an instance of “EventEmitter”

**Step 3:** Use the instance of EventEmitter class to implement the emit() method, which fires an event called “messageLogged” in the code below. The first argument in the emit() method is the event name.

**Step 4:** Use the same instance of EventEmitter class to listen to the event “messageLogged” by using the on(). The first argument in the on() is the event name and the second is the event handler of the event.

**Example:**

```
// get the reference of EventEmitter class of events module
var events = require('events');

//create an object of EventEmitter class by using above reference
var em = new events.EventEmitter();

//Subscribe for messageLogged
emitter.on('messageLogged',function()
{
    console.log("Listening to the event messageLogged")
})
emitter.emit('messageLogged')
```

**Example: Write a program for multiplication of 2 numbers using event handling in node. js. Call multiplication function as an event call. [5M]**

```
var emitter = new Events.EventEmitter();
emitter.on('mult',()=>{
    var a=5;
    var b=3;
    var c=a*b;
    console.log('Multiplication: '+c)
})
function fun_multi(){
    emitter.emit('mult')
}
fun_multi()
```

**9. URL (Uniform Resource Locator) module in node.js**

The URL module in node js is used for separating a web address into small chunks of information that is easily understandable by the user.

Since, the URL is also a built in module so we can easily import it in our javascript file by using the require method.

**Syntax:**

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

**Parsing the URL**

Let's parse a web address with the `url.parse()` method so that we can get a URL object along with its properties before breaking down our web address into small chunks inside your javascript file.

Some of the returned url object properties are:

Property	Functions
Host	Shows the name of the host/provider/server
PathName	Describes the pathname of the file where information is stored
Search	Shows the searched query as a string
Protocol	Name of the protocol used i.e. http/https
Href	Shows the complete url as a string

**Example:**

```
var url = require('url');  
var my_url = 'http://localhost:3000/index.htm?year=2020&month=May';  
//Parsing the adress:  
var que = url.parse(my_url, true);
```

//The parsing method has created an object that can be accessed using its property names.

```
console.log("Host: "+ que.host);  
console.log("PathName: "+que.pathname);  
console.log("Search Query: "+que.search);  
console.log("Protocol: "+que.protocol);  
console.log("Href: "+que.href);
```

```
//You can find about the month inside the query by storing it inside a variable and  
later accessing its properties.  
var que_info = que.query;  
console.log("Year: "+ que_info.year);
```

## 10. File System Module

To manage file operations like creating, accessing, erasing, and other similar tasks, Node.js provides a built-in module called FS (File System).

Depending on a user's needs, all file system processes can be either asynchronous or synchronous.

Use the `require()` function to incorporate the File System module:

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

The File System module is frequently used for the following purposes:

File Reading

File Creation

File Updation

File Removal

File Renaming

### a. File Reading

Use the `fs.readFile()` method to read the physical file asynchronously.

#### **Syntax:**

```
fs.readFile(fileName [,options], callback)
```

Parameter Description:

filename: Full path and name of the file as a string.

options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when readFile operation completes.

#### **Example:**

**file.js**

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

```
fs.readFile('TestFile.txt', function (err, data) {  
  if (err)  
    throw err;  
  console.log(data);  
});
```

### **Testfile.txt**

This is test file to test fs module of Node.js

## **b. Writing a File**

Use the `fs.writeFile()` method to write data to a file. If file already exists then it overwrites the existing content otherwise it creates a new file and writes data into it.

### **Syntax:**

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

Parameter Description:

filename: Full path and name of the file as a string.

Data: The content to be written in a file.

options: The options parameter can be an object or string which can include encoding, mode and flag. The default encoding is utf8 and default flag is "r".

callback: A function with two parameters err and fd. This will get called when write operation completes.

### **Example:**

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

```
fs.writeFile('test.txt', 'Hello World!', function (err) {  
  if (err)  
    console.log(err);  
  else  
    console.log('Write operation complete.');
```

```
});
```

In the same way, use the `fs.appendFile()` method to append the content to an existing file.

### **Example:**

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

```
fs.appendFile('test.txt', 'Hello World!', function (err) {  
  if (err)
```



```

        console.log(err);
        else
        console.log('Append operation complete.');
```

});

### c. Open File

Alternatively, you can open a file for reading or writing using the `fs.open()` method.

#### Syntax:

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

#### Parameter Description:

path: Full path with name of the file as a string.

Flag: The flag to perform operation

Mode: The mode for read, write or readwrite. Defaults to 0666 readwrite.

callback: A function with two parameters `err` and `fd`. This will get called when file open operation completes.

#### Flags

The following table lists all the flags which can be used in read/write operation.

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+	Like 'a+' but fails if path exists.

#### Example:

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

```
//create an empty file named mynewfile2.txt:  
  
fs.open('mynewfile2.txt', 'w', function (err, file) {  
  
  if (err) throw err;  
  
  console.log('Saved!');  
  
});
```

#### **d. Delete File**

Use the fs.unlink() method to delete an existing file.

##### **Syntax:**

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

##### **Example:**

```
var fs = require('fs');  
fs.unlink('test.txt', function () {  
  console.log('File Deleted Successfully.');
```

```
});
```