What is Node.js?

Node.js is an open-source server side runtime environment built on Chrome's V8 JavaScript engine. It provides an event driven, non-blocking (asynchronous) I/O and cross-platform runtime environment for building highly scalable server-side application using JavaScript.

Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc. However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.

Node.js was written and introduced by Ryan Dahl in 2009.

Advantages of Node.js

1. Node.js is an open-source framework under MIT license. (MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT).)
2. Uses JavaScript to build entire server side application.
3. Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.
4. Asynchronous by default. So it performs faster than other frameworks.
5. Cross-platform framework that runs on Windows, MAC or Linux

Install Node.js

Node.js development environment can be setup in Windows, Mac, Linux and Solaris. The following tools/SDK are required for developing a Node.js application on any platform.

1. Node.js
2. Node Package Manager (NPM)
3. IDE (Integrated Development Environment) or TextEditor

NPM (Node Package Manager) is included in Node.js installation so there is no need to install it separately.

-Visit Node.js official web site https://nodejs.org. It will automatically detect OS and display download link as per your Operating System.

-Download the installer for windows by clicking on LTS or Current version button. Here, we will install the latest version LTS for windows that has long time support. However, you can also install the Current version which will have the latest features.

-Click Next to read and accept the License Agreement and then click Install. It will install Node.js quickly on your computer. Finally, click finish to complete the installation.

Verify Installation

Once you install Node.js on your computer, you can verify it by opening the command prompt and typing node -v. If Node.js is installed successfully then it will display the version of the Node.js installed on your machine

Node.js Module

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 implements CommonJS modules standard. CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

Node.js Module Types

Node.js includes three types of modules:

1. Core Modules
2. Local Modules
3. Third Party Modules

1. Node.js Core Modules

Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary

distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

The following table lists some of the important core modules in Node.js.

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

1. **Node.js HTTP Module**

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

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

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

The following example demonstrates how to use Node.js http module to create a web server.

Example: Load and Use Core http Module

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

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

```
  //write code here

});

server.listen(5000);
```

In the above example, require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().In this way, you can load and use Node.js core modules in your application.

2. Node.js File System Module

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

To include the File System module, use the require() method:

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

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Read Files

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

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

demofile1.html

```
<html>
<body>
<h1>My Header</h1>
```

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

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

Example

```
var http = require('http');
var fs = require('fs');
http.createServer(function (req, res) {
  fs.readFile('demofile1.html', function(err, data) {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

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

Create Files

The File System module has methods for creating new files:

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

The fs.appendFile() method appends specified content to a file. If the file does not exist, the file will be created:

Example

Create a new file using the appendFile() method:

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

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

```
  console.log('Saved!');
});
```

The fs.open() method takes a "flag" as the second argument, if the flag is "w" for "writing", the specified file is opened for writing. If the file does not exist, an empty file is created:

Example

Create a new, empty file using the open() method:

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

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

The fs.writeFile() method replaces the specified file and content if it exists. If the file does not exist, a new file, containing the specified content, will be created:

Example

Create a new file using the writeFile() method:

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

fs.writeFile('mynewfile3.txt', 'Hello content!', function (err) {
  if (err) throw err;
  console.log('Saved!');
});
```

Update Files

The File System module has methods for updating files:

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

The fs.appendFile() method appends the specified content at the end of the specified file:

Example

Append "This is my text." to the end of the file "mynewfile1.txt":

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

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

The fs.writeFile() method replaces the specified file and content:

Example

Replace the content of the file "mynewfile3.txt":

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

fs.writeFile('mynewfile3.txt', 'This is my text', function (err) {
  if (err) throw err;
  console.log('Replaced!');
});
```

Delete Files

To delete a file with the File System module, use the fs.unlink() method.

The fs.unlink() method deletes the specified file:

Example

Delete "mynewfile2.txt":

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

fs.unlink('mynewfile2.txt', function (err) {
```

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

Rename Files

To rename a file with the File System module, use the fs.rename() method.

The fs.rename() method renames the specified file:

Example

Rename "mynewfile1.txt" to "myrenamedfile.txt":

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

fs.rename('mynewfile1.txt', 'myrenamedfile.txt', function (err) {
  if (err) throw err;
  console.log('File Renamed!');
});
```

3. Node.js URL Module

The URL module splits up a web address into readable parts.

To include the URL module, use the require() method:

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

Parse an address with the url.parse() method, and it will return a URL object with each part of the address as properties:

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2017&month=february';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2017&month=february'
```

```
var qdata = q.query; //returns an object: { year: 2017, month: 'february' }
console.log(qdata.month); //returns 'february'
```

2. Node.js Local Module

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it. For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Writing Simple Module

Let's write simple logging module which logs the information, warning or error to the console.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

Log.js

```
var log = {
        info: function (info) {
           console.log('Info: ' + info);
        },
        warning:function (warning) {
           console.log('Warning: ' + warning);
        },
        error:function (error) {
           console.log('Error: ' + error);
        }
   };

module.exports = log
```

In the above example of logging module, we have created an object with three functions - info(), warning() and error(). At the end, we have assigned this object to **module.exports**. The module.exports in the above example exposes a log object as a module.

The *module.exports* is a special object which is included in every JS file in the Node.js application by default. Use **module.exports** or **exports** to expose a function, object or variable as a module in Node.js.

Loading Local Module

To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

The following example demonstrates how to use the above logging module contained in Log.js.

app.js

```
var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');
```

In the above example, app.js is using log module. First, it loads the logging module using require() function and specified path where logging module is stored. Logging module is contained in Log.js file in the root folder. So, we have specified the path './Log.js' in the require() function. The '.' denotes a root folder.

The require() function returns a log object because logging module exposes an object in Log.js using module.exports. So now you can use logging module as an object and call any of its function using dot notation e.g myLogModule.info() or myLogModule.warning() or myLogModule.error()

Run the above example using command prompt (in Windows) as shown below.

```
C:\>nodeapp.js
Info: Node.js started
```

Thus, you can create a local module using module.exports and use it in your application.

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

NPM - Node Package Manager

Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application. It is also an online repository for open-source Node.js packages. The node community around the world creates useful modules and publishes them as packages in this repository.
NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt.

```
C:\> npm –v
```

If you have an older version of NPM then you can update it to the latest version using the following command.

```
C:\> npm install npm –g
```

To access NPM help, write **npm help** in the command prompt or terminal window.

```
C:\> npm help
```

NPM performs the operation in two modes: global and local. 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.

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. The above command will create ExpressJS folder under node_modules folder in the root folder of your project and install Express.js there.

Add Dependency into package.json

Use --save at the end of the install command to add dependency entry into package.json of your application.

For example, the following command will install ExpressJS in your application and also adds dependency entry into the package.json.

```
C:\MyNodeProj> npm install express –save
```

The package.json of NodejsConsoleApp project will look something like below.

package.json

```
{
  "name": "NodejsConsoleApp",
  "version": "0.0.0",
  "description": "NodejsConsoleApp",
  "main": "app.js",
  "author": {
    "name": "Dev",
    "email": ""
  },
  "dependencies": {
    "express": "^4.13.3"
  }
}
```

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
```

Update Package

To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder and write the following update command.

```
C:\MyNodeProj> npm update <package name>
```

The following command will update the existing ExpressJS module to the latest version.

```
C:\MyNodeProj> npm update express
```

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
```