What is Node JS

Node.js is an open-source, cross-platform runtime environment used for development of server-side web applications. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.

Node.js is based on an event-driven architecture and a non-blocking Input/Output API that is designed to optimize an application's throughput and scalability for real-time web applications.

Over a long period of time, the framework available for web development were all based on a stateless model. A stateless model is where the data generated in one session (such as information about user settings and events that occurred) is not maintained for usage in the next session with that user.

A lot of work had to be done to maintain the session information between requests for a user. But with Node.js there is finally a way for web applications to have a real-time, two-way connections, where both the client and server can initiate communication, allowing them to exchange data freely.

Why use Node.js?

We will have a look into the real worth of Node.js in the coming chapters, but what is it that makes this framework so famous. Over the years, most of the applications were based on a stateless request-response framework. In these sort of applications, it is up to the developer to ensure the right code was put in place to ensure the state of web session was maintained while the user was working with the system.

But with Node.js web applications, you can now work in real-time and have a 2-way communication. The state is maintained, and the either the client or server can start the communication.

Features of Node.js

Let's look at some of the key features of Node.js

Asynchronous event driven IO helps concurrent request handling – This is probably the biggest selling points of Node.js. This feature basically means that if a request is received by Node for some Input/Output operation, it will execute the operation in the background and continue with processing other requests.

This is quite different from other programming languages. A simple example of this is given in the code below

```
var fs = require('fs');
    fs.readFile("Sample.txt",function(error,data)
    {
        console.log("Reading Data completed");
  });
```

The above code snippet looks at reading a file called Sample.txt. In other programming languages, the next line of processing would only happen once the entire file is read.

But in the case of Node.js the important fraction of code to notice is the declaration of the function ('function(error,data)'). This is known as a callback function.

So what happens here is that the file reading operation will start in the background. And other processing can happen simultaneously while the file is being read. Once the file read operation is completed, this anonymous function will be called and the text "Reading Data completed" will be written to the console log.

Node uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node also become faster.

Handling of concurrent requests – Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process.

The Node.js library used JavaScript – This is another important aspect of development in Node.js. A major part of the development community are already well versed in javascript, and hence, development in Node.js becomes easier for a developer who knows javascript.

There are an Active and vibrant community for the Node.js framework. Because of the active community, there are always keys updates made available to the framework. This helps to keep the framework always up-to-date with the latest trends in web development.

Who uses Node.js

Node.js is used by a variety of large companies. Below is a list of a few of them.

1) Paypal – A lot of sites within Paypal have also started the transition onto Node.js.
2) LinkedIn - LinkedIn is using Node.js to power their Mobile Servers, which powers the iPhone, Android, and Mobile Web products.
3) Mozilla has implemented Node.js to support browser APIs which has half a billion installs.
4) Ebay hosts their HTTP API service in Node.js

When to Use Node.js

Node.js is best for usage in streaming or event-based real-time applications like

Chat applications

Game servers – Fast and high-performance servers that need to processes thousands of requests at a time, then this is an ideal framework.

Good for collaborative environment – This is good for environments which manage document. In document management environment you will have multiple people who post their documents and do constant changes by checking out and checking in documents. So Node.js is good for these environments because the event loop in Node.js can be triggered whenever documents are changed in a document managed environment.

Advertisement servers – Again here you could have thousands of request to pull advertisements from the central server and Node.js can be an ideal framework to handle this.

Streaming servers – Another ideal scenario to use Node is for multimedia streaming servers wherein clients have request's to pull different multimedia contents from this server.

Node.js is good when you need high levels of concurrency but less amount of dedicated CPU time.

Best of all, since Node.js is built on javascript, it's best suited when you build client-side applications which are based on the same javascript framework.

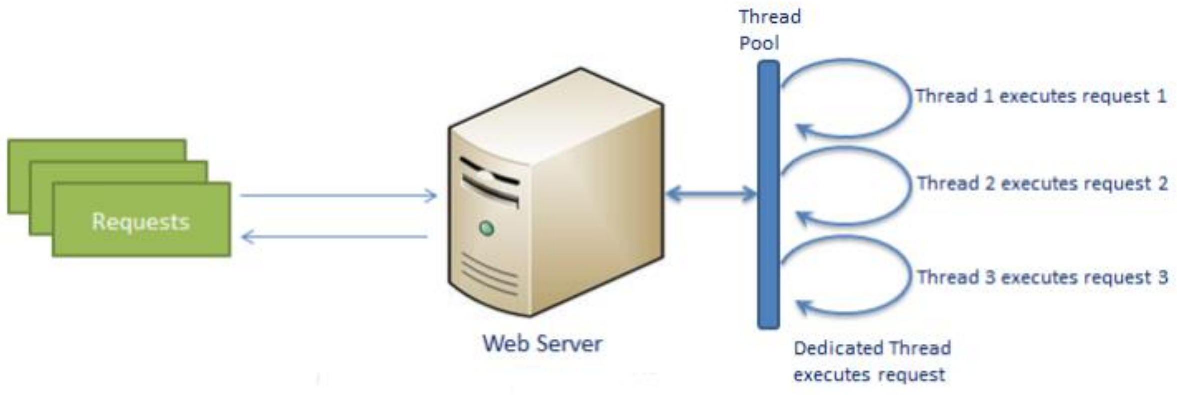
When to not use Node.js

Node.js can be used for a lot of applications with various purpose, the only scenario where it should not be used is if there are long processing times which is required by the application.

Node is structured to be single threaded. If any application is required to carry out some long running calculations in the background. So if the server is doing some calculation, it won't be able to process any other requests. As discussed above, Node.js is best when processing needs less dedicated CPU time.

Traditional Web Server Model

In the traditional web server model, each request is handled by a dedicated thread from the thread pool. If no thread is available in the thread pool at any point of time then the request waits till the next available thread. Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.

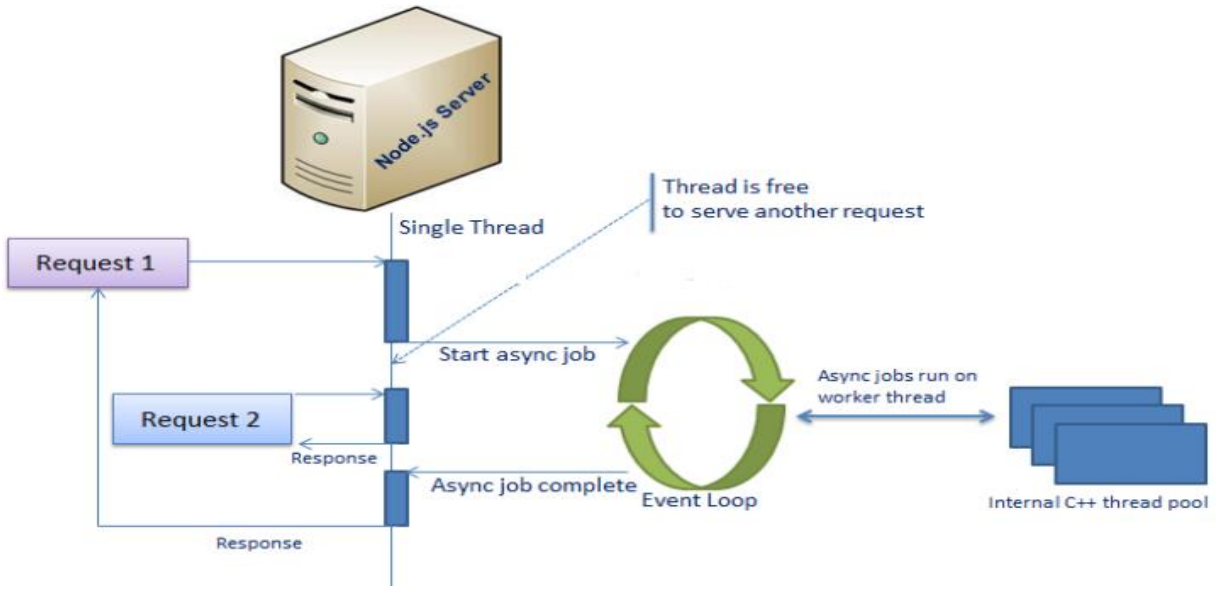

Thread Pool
Requests
Web Server
Thread 1 executes request 1
Thread 2 executes request 2
Thread 3 executes request 3
Dedicated Thread executes request

Node.js Process Model

Node.js processes user requests differently when compared to a traditional web server model. Node.js runs in a single process and the application code runs in a single thread and thereby needs less resources than other platforms. All the user requests to your web application will be handled by a single thread and all the I/O work or long running job is performed asynchronously for a particular request. So, this single thread doesn't have to wait for the request to complete and is free to handle the next request. When asynchronous I/O work completes then it processes the request further and sends the response.

An event loop is constantly watching for the events to be raised for an asynchronous job and executing callback function when the job completes. Internally, Node.js uses libevfor the event loop which in turn uses internal C++ thread pool to provide asynchronous I/O.

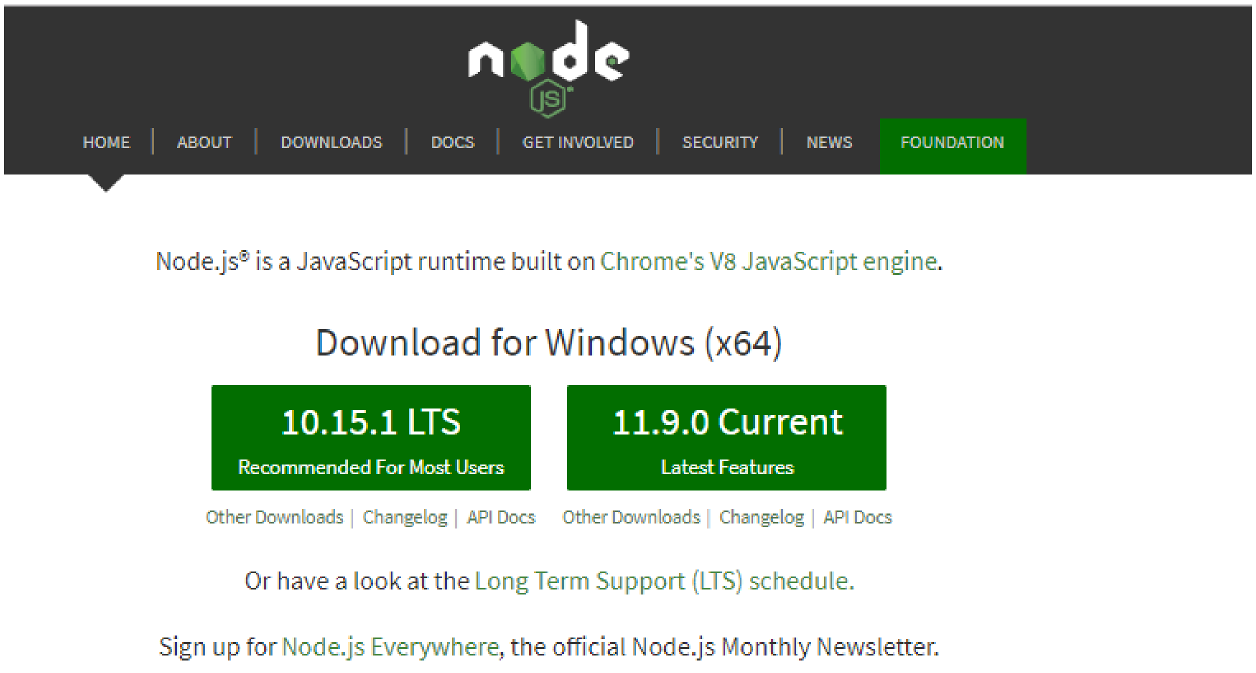
The following figure illustrates asynchronous web server model using Node.js.



Node.js process model increases the performance and scalability with a few caveats. Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Node JS Installation

We can install the node JS from the official website https://nodejs.org/en/



Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine.

Download for Windows (x64)

10.15.1 LTS
Recommended For Most Users

11.9.0 Current
Latest Features

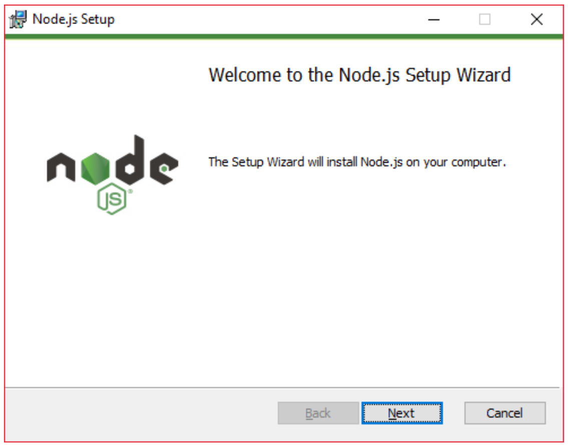
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

Or have a look at the Long Term Support (LTS) schedule.

Sign up for Node.js Everywhere, the official Node.js Monthly Newsletter.

Choose any installable version, it downloads a '.msi' file in your system. For Mac Users click on other downloads option below each version.

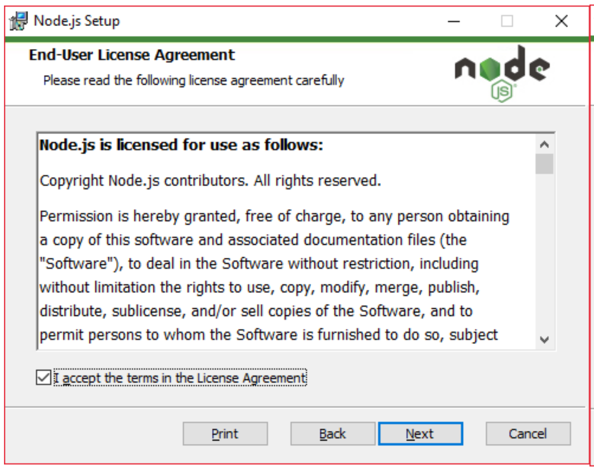
Now install the downloaded 'node.10.15.1.msi' file.



Welcome to the Node.js Setup Wizard

The Setup Wizard will install Node.js on your computer.

Click on Next Button , and accept the licence.


Node.js Setup

End-User License Agreement

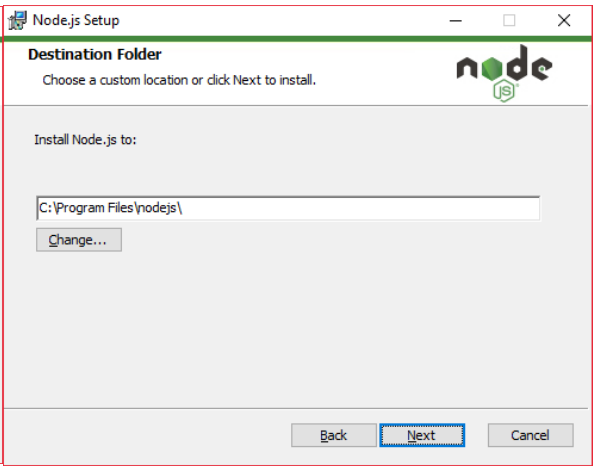
Please read the following license agreement carefully

Node.js is licensed for use as follows:

Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

[✓] I accept the terms in the License Agreement


Node.js Setup

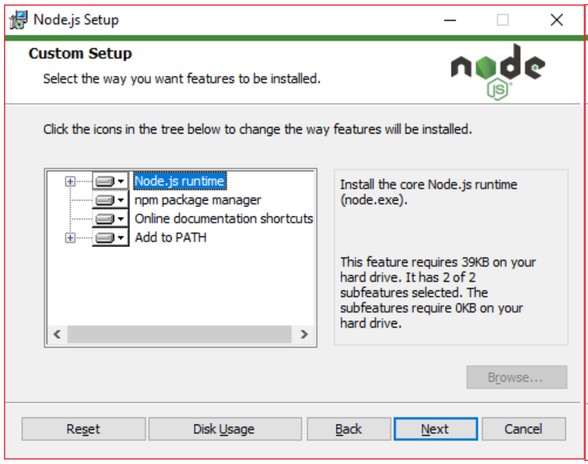
Destination Folder

Choose a custom location or click Next to install.

Install Node.js to:

C:\Program Files\nodejs\

Select the Destination path, keep the default program files location.


Node.js Setup

Custom Setup

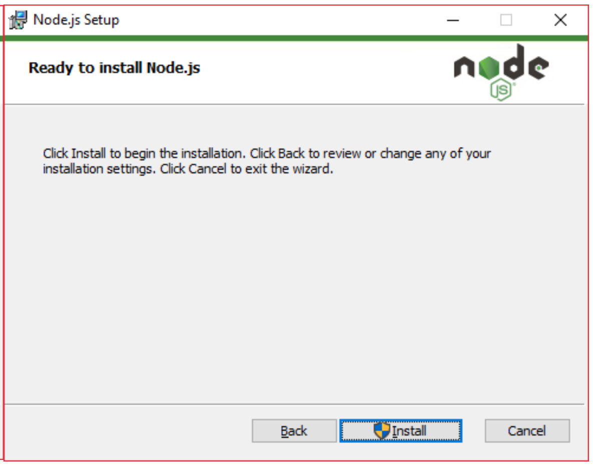
Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

- Node.js runtime
- npm package manager
- Online documentation shortcuts
- Add to PATH

Install the core Node.js runtime (node.exe).

This feature requires 39KB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 0KB on your hard drive.

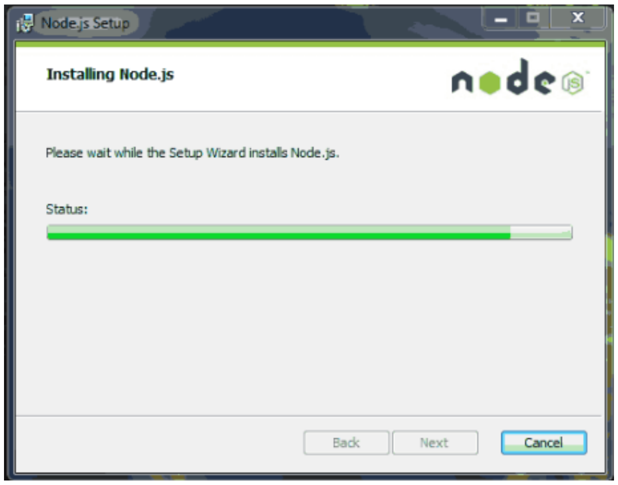

Node.js Setup

Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

No changes in the custom settings , just click on next

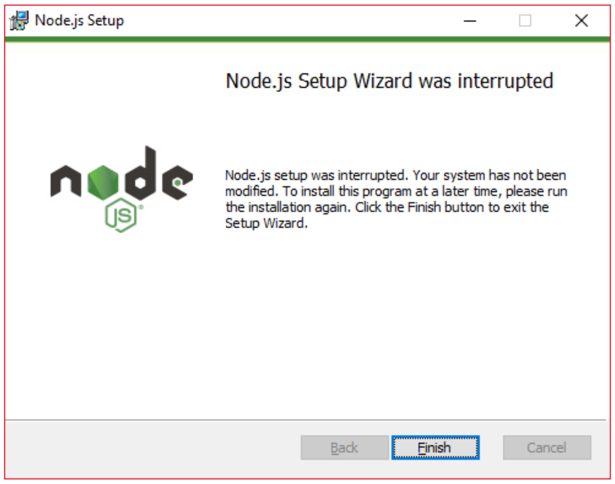
Click on Install button, it installs the node js in your machine.


Node.js Setup

Installing Node.js

Please wait while the Setup Wizard installs Node.js.

Status:


Node.js Setup

Node.js Setup Wizard was interrupted

Node.js setup was interrupted. Your system has not been modified. To install this program at a later time, please run the installation again. Click the Finish button to exit the Setup Wizard.

Now we can verify the node JS version in the command prompt.

```
C:\Users\NAVEEN SAGGAM>node -v
v10.14.2

C:\Users\NAVEEN SAGGAM>
```

Node.js Console - REPL

Node.js comes with virtual environment called REPL (aka Node shell). REPL stands for Read-Eval-Print-Loop. It is a quick and easy way to test simple Node.js/JavaScript code.

REPL stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode. Node.js or Node comes bundled with a REPL environment. It performs the following tasks:

● Read - Reads user's input, parses 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 the user presses ctrl-c twice. The REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes

To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux) and type node as shown below. It will change the prompt to > in Windows and MAC.

```
C:\Users\NAVEEN SAGGAM>node
>
```

Simple Expressions

```
C:\Users\NAVEEN SAGGAM>node
> 10 + 20
30
>
```

Use variables

You can make use variables to store values and print later like any conventional script. If var keyword is not used, then the value is stored in the variable and printed. Whereas if var keyword is used, then the value is stored but not printed. You can print variables using console.log().

```
C:\Users\NAVEEN SAGGAM>node
> let x = 10;
undefined
> let y = 20;
undefined
> console.log(`The Sum of x , y is : ${x + y}`);
The Sum of x , y is : 30
undefined
>
```

Multiline Expression

Node REPL supports multiline expression similar to JavaScript. Let's check the following dowhile loop in action:

```
C:\Users\NAVEEN SAGGAM>node
> let number = 10;
undefined
> let output = '';
undefined
> for(let i=0; i<=number; i++){
... output += `${i} `;
... }
'0 1 2 3 4 5 6 7 8 9 10 '
>
```

REPL Commands

- ctrl + c - terminate the current command.
- ctrl + c twice - terminate the Node REPL.
- ctrl + d - terminate the Node REPL.
- Up/Down Keys - see command history and modify previous commands.
- tab Keys - list of current commands.
- .help - list of all commands.
- .break - exit from multiline expression.
- .clear - exit from multiline expression.
- .save filename - save the current Node REPL session to a file.
- .load filename - load file content in current Node REPL session.

Node.js Basics

Node.js supports JavaScript. So, JavaScript syntax on Node.js is similar to the browser's JavaScript syntax.

Primitive Types

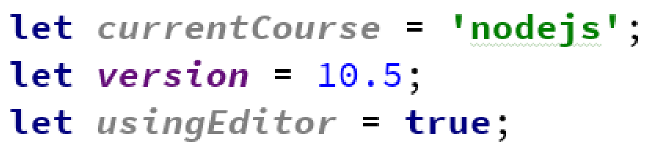
Node.js includes following primitive types:

1. String
2. Number
3. Boolean
4. Undefined
5. Null

Everything else is an object in Node.js.

Loose Typing

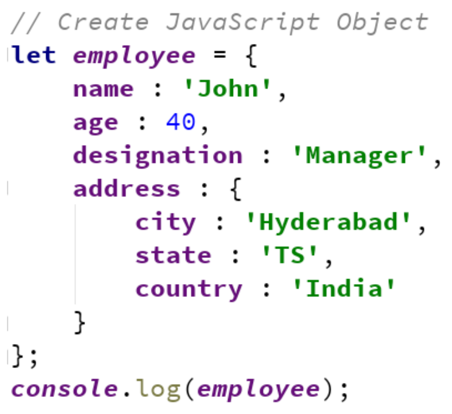
JavaScript in Node.js supports loose typing like the browser's JavaScript. Use let keyword to declare a variable of any type.



```
let currentCourse = 'nodejs';
let version = 10.5;
let usingEditor = true;
```

Object Literal

Object literal syntax is same as browser's JavaScript.



```
// Create JavaScript Object
let employee = {
    name : 'John',
    age : 40,
    designation : 'Manager',
    address : {
        city : 'Hyderabad',
        state : 'TS',
        country : 'India'
    }
};
console.log(employee);
```

Functions

Functions are first class citizens in Node's JavaScript, similar to the browser's JavaScript. A function can have attributes and properties also. It can be treated like a class in JavaScript.

```
// Normal Function
function printEmployee_1(employee) {
    console.log(employee);
}

// Function Expression
let printEmployee_2 = function(employee) {
    console.log(employee);
};

// Arrow Function from ES6
let printEmployee_3 = (employee) => {
    console.log(employee);
};
```

Buffer

Node.js includes an additional data type called Buffer (not available in browser's JavaScript). Buffer is mainly used to store binary data, while reading from a file or receiving packets over the network.

process object

Each Node.js script runs in a process. It includes process object to get all the information about the current process of Node.js application.

The following example shows how to get process information in REPL using process object.

Defaults to local

Node's JavaScript is different from browser's JavaScript when it comes to global scope. In the browser's JavaScript, variables declared without var keyword become global. In Node.js, everything becomes local by default.

Access Global Scope

In a browser, global scope is the window object. In Node.js, global object represents the global scope.

To add something in global scope, you need to export it using export or module.export. The same way, import modules/object using require() function to access it from the global scope.

For example, to export an object in Node.js, use exports.name = object.

Now, you can import log object using require() function and use it anywhere in your Node.js project.

Learn about modules in detail in the next section.

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.

Node.js Module Types

Node.js includes three types of modules:

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

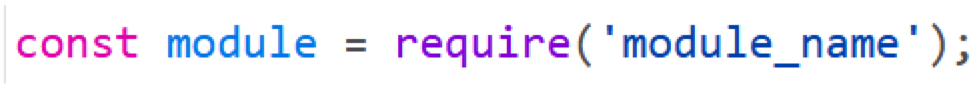
Node.js Core Modules

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.
https	Same as http module , with secured SSL server creations
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.
os	Os module deals with accessing the underlying operating system properties.

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



```
const 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

```
const os = require('os'); // OS module

let totalMem = os.totalmem();
console.log(`Total Memory : ${totalMem}`);

let freeMem = os.freemem();
console.log(`Free Memory : ${freeMem}`);

let hostName = os.hostname();
console.log(`HostName : ${hostName}`);

let userInfo = os.userInfo().username;
console.log(`UserName : ${userInfo}`);
```

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.

Node.js custom / Local Modules

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.

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.

```
let 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 // to export the module outside
};
```

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.

Now, let's see how to use the above logging module in our application.

```
const log = require('./log');

log.info('This is an info message');
log.warning('This is an info message');
log.error('This is an info message');
```

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.

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' or './log' 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 log.info() or log.warning() or log.error()

Export Module in Node.js

In the previous section, you learned how to write a local module using module.exports. In this section, you will learn how to expose different types as a module using module.exports.

The module.exports or exports is a special object which is included in every JS file in the Node.js application by default. module is a variable that represents current module and exports is an object that will be exposed as a module. So, whatever you assign to module.exports or exports, will be exposed as a module.

Let's see how to expose different types as a module using module.exports.

Export Literals

As mentioned above, exports is an object. So it exposes whatever you assigned to it as a module. For example, if you assign a string literal then it will expose that string literal as a module.

The following example exposes simple string message as a module in Message.js.

```
module.exports = 'Good Morning';
```

Now, import this message module and use it as shown below.

```
const msg = require('./messages');
console.log(msg);
```

Note: You must specify './' as a path of root folder to import a local module. However, you do not need to specify path to import Node.js core module or NPM module in the require() function.

Export Object

exports is an object. So, you can attach properties or methods to it. The following example exposes an object with a string property in Message.js file.

```
exports.SimpleMessage = 'Hello world';
(or)
module.exports.SimpleMessage = 'Hello world';
```

In the above example, we have attached a property "SimpleMessage" to the exports object. Now, import and use this module as shown below.

```
// app.js
const msg = require('./Messages.js');
console.log(msg.SimpleMessage);
```

The same way as above, you can expose an object with function. The following example exposes an object with log function as a module.

```
// log.js
module.exports.log = function (msg) {
    console.log(msg);
};
```

The above module will expose an object

```
// app.js
const msg = require('./log.js');
msg.log('Hello World');
```

You can also attach an object to module.exports as shown below.

```
// data.js
module.exports = {
    firstName: 'John',
    lastName: 'Wilson'
}
```

```
// app.js
const person = require('./data.js');
console.log(person.firstName + ' ' + person.lastName);
```

Export Function

You can attach an anonymous function to exports object as shown below.

```
// log.js
module.exports = function (msg) {
    console.log(msg);
};
```

Now, you can use the above module as below.

```
// app.js
const msg = require('./Log.js');
msg('Hello World');
```

The msg variable becomes function expression in the above example. So, you can invoke the function using parenthesis (). Run the above example and see the output as shown below.

Load Module from Separate Folder

Use the full path of a module file where you have exported it using module.exports. For example, if log module in the log.js is stored under "utility" folder under the root folder of your application then import it as shown below.

```
// app.js
var log = require('./utility/log.js');
```

In the above example, . is for root folder and then specify exact path of your module file. Node.js also allows us to specify the path to the folder without specifying file name.

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..

Official website: https://www.npmjs.com

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:\Users>node -v
v10.15.1
```

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

```
C:\Users>npm i npm -g
```

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

```
C:\Users>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:\Users>npm install <package_name>
```

For example, the following command will install ExpressJS into Project1 folder.

```
C:\project_1>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:\project_1>npm install express --save
```

The package.json of Node JS project will look something like below.

```
// package.json
{
  "name": "express_app",
  "version": "1.0.0",
  "description": "",
  "main": "app.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.16.4"
  }
}
```

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:\project_1>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:\project_1>npm update <package_name>
```

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

```
C:\project_1>npm update express
```

Uninstall Packages

Use the following command to remove a local package from your project.

```
C:\project_1>npm uninstall <package_name>
```

The following command will uninstall ExpressJS from the application.

```
C:\project_1>npm uninstall express
```