**How to Download & Install Node.js - NPM on Windows**

To start building your Node.js applications, the first step is the installation of the node.js framework. The Node.js framework is available for a variety of operating systems right from Windows to Ubuntu and OS X. Once the Node.js framework is installed, you can start building your first Node.js applications.

Node.js also has the ability to embedded external functionality or extended functionality by making use of custom modules. These modules have to be installed separately. An example of a module is the[MongoDB](https://www.guru99.com/mongodb-tutorials.html)module which allows you to work with MongoDB databases from your Node.js application.

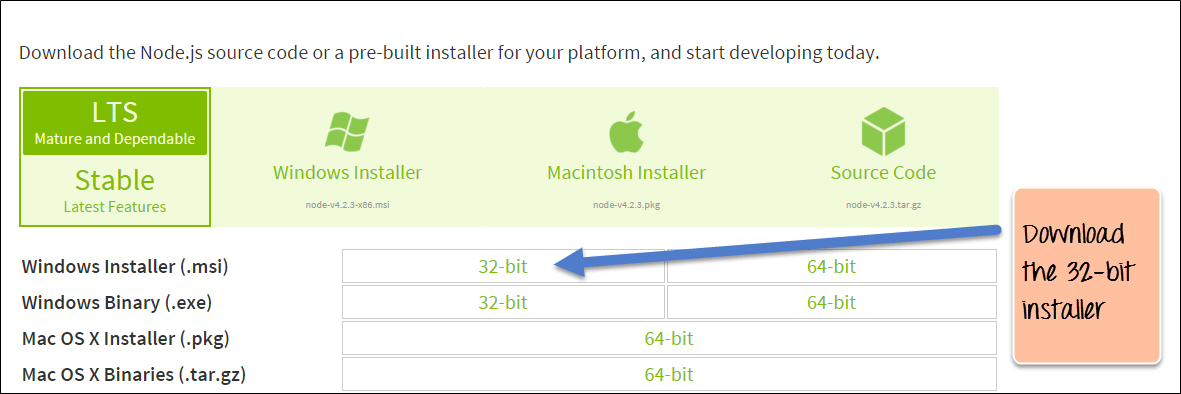
In this tutorial, you will learn-

* [How to install Node.js on Windows](https://www.guru99.com/download-install-node-js.html#1)
* [Installing NPM (Node Package Manager) on Windows](https://www.guru99.com/download-install-node-js.html#2)
* [Running your first Hello World application in Node.js](https://www.guru99.com/download-install-node-js.html#3)

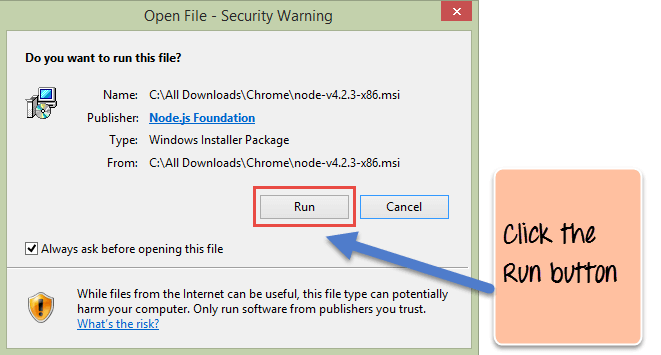
**How to install Node.js on Windows**

The first steps in using Node.js is the installation of the Node.js libraries on the client system. To perform the installation of Node.js, perform the below steps;

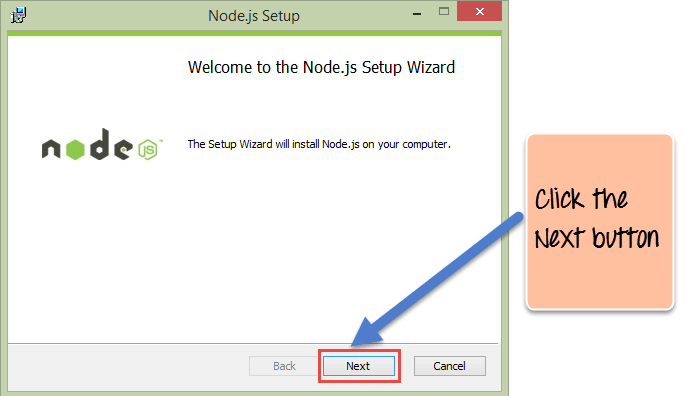
**Step 1)** Go to the site <https://nodejs.org/en/download/> and download the necessary binary files. In our example, we are going to download the 32-bit setup files for Node.js.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns1.png)

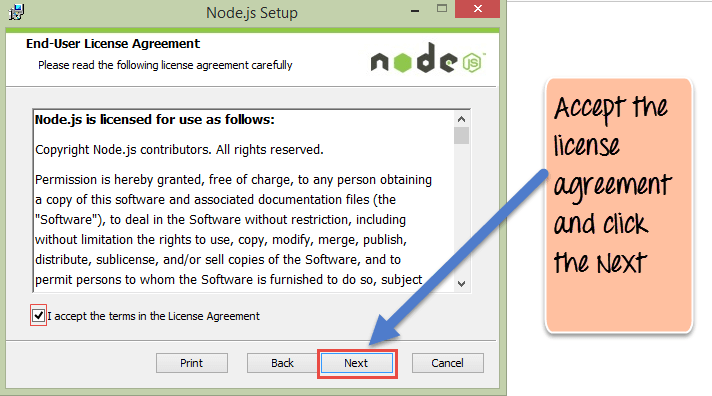
**Step 2)** Double click on the downloaded .msi file to start the installation. Click the Run button on the first screen to begin the installation.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns2.png)

**Step 3)** In the next screen, click the "Next" button to continue with the installation

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns3.png)

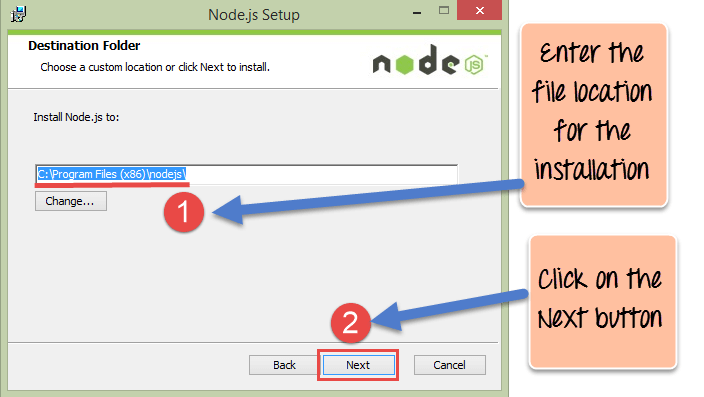
**Step 4)** In the next screen, Accept the license agreement and click on the Next button.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns4.png)

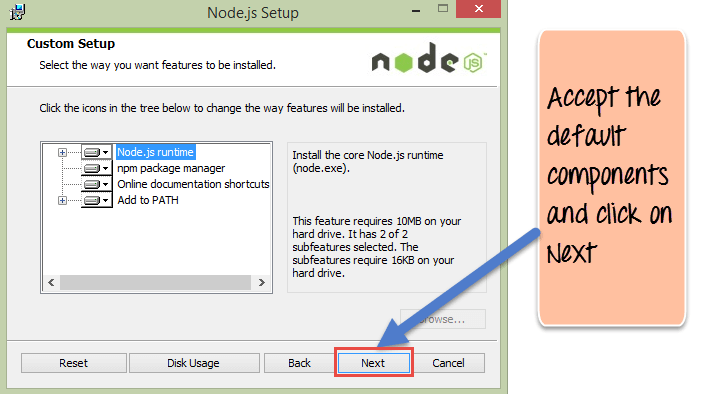
**Step 5)** In the next screen, choose the location where Node.js needs to be installed and then click on the Next button.

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.

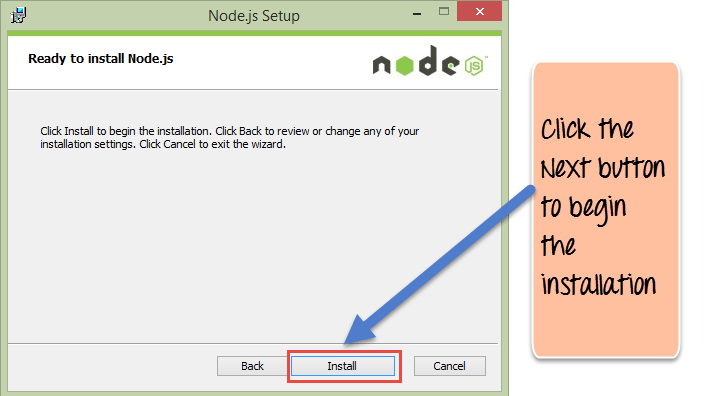
2. Click on the Next button to proceed ahead with the installation.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns5.png)

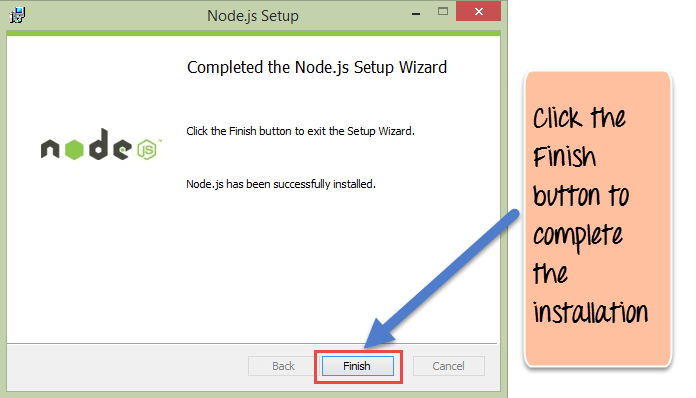
**Step 6)** Accept the default components and click on the Next button.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns6.png)

**Step 7)**In the next screen, click the Install button to start the installation.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns7.png)

**Step 8)** Click the Finish button to complete the installation.

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns8.png)

Windows is now recommending that developers use Node.js with [WSL2](https://docs.microsoft.com/en-us/windows/nodejs/setup-on-wsl2)(the Windows subsystem for Linux)

**Installing NPM (Node Package Manager) on Windows**

The other way to install Node.js on any client machine is to use a "package manager."

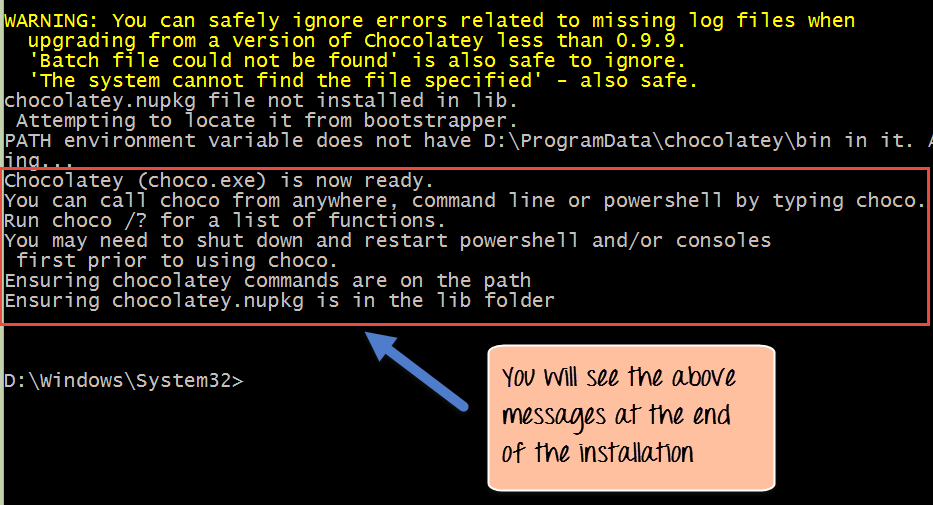
On windows, the node package manager is known as Chocolatey. It was designed to be a decentralized framework for quickly installing applications and tools that you need.

To install Node.js via Chocolatey, the following steps need to be performed.

**Step 1)** Installing Chocolatey – The Chocolatey website (<https://chocolatey.org/>) has very clear instructions on how this framework needs to be installed.

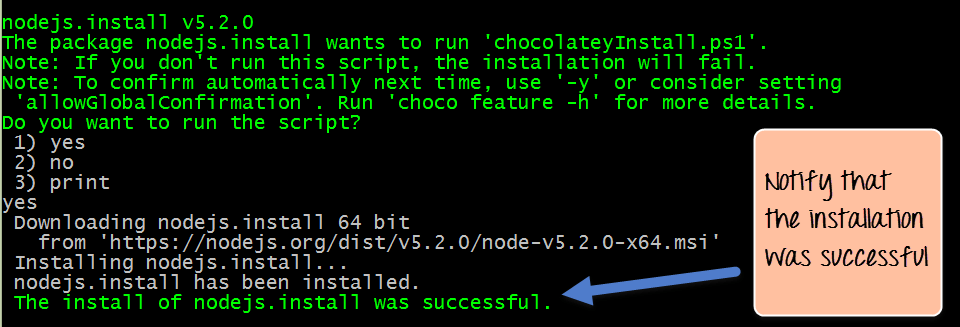
* The first step is to run the below command in the command prompt windows. This command is taken from the Chocolatey web site and is the standard command for installing Node.js via Chocolatey.
* The below command is a PowerShell command which calls the remote PowerShell script on the Chocolatey website. This command needs to be run in a PowerShell command window.
* This PowerShell script does all the necessary work of downloading the required components and installing them accordingly.

@powershell -NoProfile -ExecutionPolicy Bypass -Command "iex ((new-object wet.webclient).DownloadString('https://chocolatey.org/install.ps1'))" && SET PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns9.png)

**Step 2)** The next step is to install Node.js to your local machine using the Chocolatey, package manager. This can be done by running the below command in the command prompt.

**cinst nodejs install**

[](https://www.guru99.com/images/NodeJS/010716_0458_DownloadIns10.png)

If the installation is successful, you will get the message of the successful installation of Node.js.

**Note:**If you get an error like "C:\ProgramData\chocolatey\lib\libreoffice\tools\chocolateyInstall.ps1" Then manually create the folder in the path

**Running your first Hello World application in Node.js**

Once you have downloaded and installed Node.js on your computer, let's try to display "Hello World" in a web browser.

Create file Node.js with file name firstprogram.js

var http = require('http');

http.createServer(function (req, res) {

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

res.end('Hello World!');

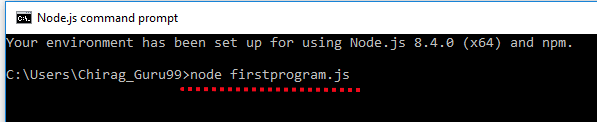
}).listen(8080);

**Code Explanation:**

1. The basic functionality of the "require" function is that it reads a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)file, executes the file, and then proceeds to return an object. Using this object, one can then use the various functionalities available in the module called by the require function. So in our case, since we want to use the functionality of HTTP and we are using the require(http) command.
2. In this 2nd line of code, we are creating a server application which is based on a simple function. This function is called, whenever a request is made to our server application.
3. When a request is received, we are asking our function to return a "Hello World" response to the client. The writeHead function is used to send header data to the client, and while the end function will close the connection to the client.
4. We are then using the server.listen function to make our server application listen to client requests on port no 8080. You can specify any available port over here.

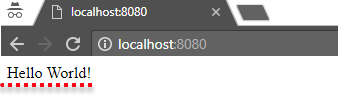
**Executing the code**

* 1. Save the file on your computer: C:\Users\Your Name\ firstprogram.js
  2. In the command prompt, navigate to the folder where the file is stored. Enter the command Node firstprogram.js

[](https://www.guru99.com/images/NodeJS/Helloword_nodejs_v1.png)

* 1. Now, your computer works as a server! If anyone tries to access your computer on port 8080, they will get a "Hello World!" message in return!
  2. Start your internet browser, and type in the address: http://localhost:8080

**OutPut**

[](https://www.guru99.com/images/NodeJS/Helloword_nodejs_v2.png)

**Summary**

* We have seen the installation of Node.js via the MSI installation module, which is available on the Node.js website. This installation installs the necessary modules which are required to run a Node.js application on the client.
* Node.js can also be installed via a package manager. The package manager for windows is known as Chocolatey. By running some simple commands in the command prompt, the Chocolatey package manager automatically downloads the necessary files and then installs them on the client machine.
* A simple Node.js application consists of creating a server which listens on a particular port. When a request comes to the server, the server automatically sends a 'Hello World' response to the client.

**Node.js NPM Tutorial: Create, Publish, Extend & Manage**

A module in Node.js is a logical encapsulation of code in a single unit. It's always a good programming practice to always segregate code in such a way that makes it more manageable and maintainable for future purposes. That's where modules in Node.js comes in action.

Since each module is an independent entity with its own encapsulated functionality, it can be managed as a separate unit of work.

In this tutorial, you will learn-

* [What are modules in Node.js](https://www.guru99.com/node-js-modules-create-publish.html#1)
* [Using modules in Node.js](https://www.guru99.com/node-js-modules-create-publish.html#2)
* [Creating NPM modules](https://www.guru99.com/node-js-modules-create-publish.html#3)
* [Extending modules](https://www.guru99.com/node-js-modules-create-publish.html#4)
* [Publishing NPM Modules](https://www.guru99.com/node-js-modules-create-publish.html#5)
* [Managing third-party packages with npm](https://www.guru99.com/node-js-modules-create-publish.html#6)
* [What is the package.json file](https://www.guru99.com/node-js-modules-create-publish.html#7)

**What are modules in Node.js?**

As stated earlier, modules in Node js are a way of encapsulating code in a separate logical unit. There are many readymade modules available in the market which can be used within Node js.

Below are some of the popular modules which are used in a Node js application

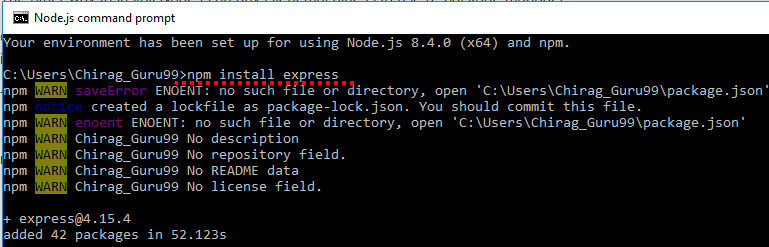
1. **Express framework** – Express is a minimal and flexible Node js web application framework that provides a robust set of features for the web and[mobile](https://www.guru99.com/mobile-testing.html)applications.
2. **Socket.io** - Socket.IO enables real-time bidirectional event-based communication. This module is good for creation of chatting based applications.
3. **Jade**- Jade is a high-performance template engine and implemented with[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)for node and browsers.
4. **MongoDB** - The[MongoDB](https://www.guru99.com/mongodb-tutorials.html)Node.js driver is the officially supported node.js driver for MongoDB.
5. **Restify** - restify is a lightweight framework, similar to express for building REST APIs
6. **Bluebird**- Bluebird is a fully-featured promise library with a focus on innovative features and performance

**Using modules in Node.js**

In order to use modules in a Node.js application, they first need to be installed using the Node package manager.

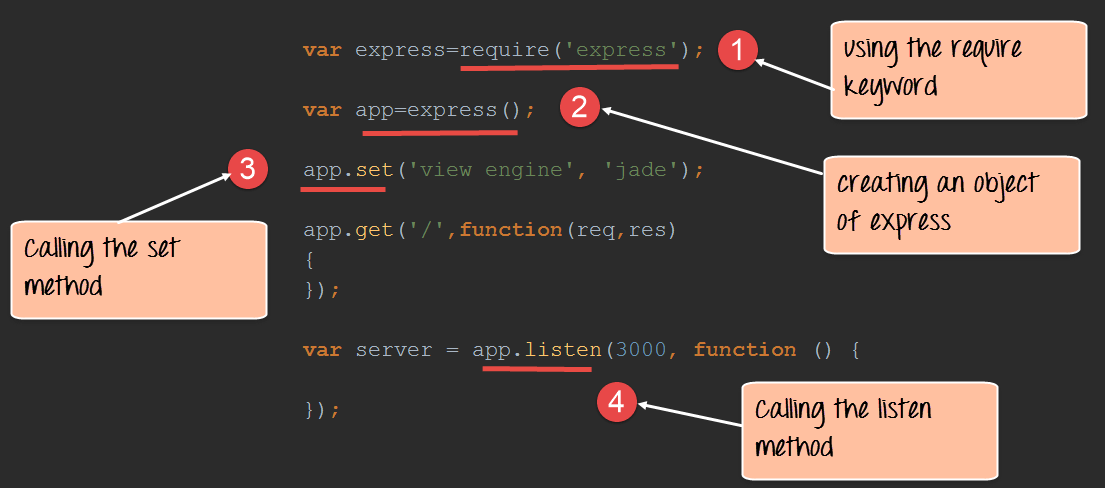
The below command line shows how a module "express" can be installed.

**npm install express**

[](https://www.guru99.com/images/NodeJS/npm_install_express.png)

* The above command will download the necessary files which contain the "express modules" and take care of the installation as well
* Once the module has been installed, in order to use a module in a Node.js application, you need to use the 'require' keyword. This keyword is a way that Node.js uses to incorporate the functionality of a module in an application.

Let's look at an example of how we can use the "require" keyword. The below - code example shows how to use the require function

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul1.png)

var express=require('express');

var app=express();

app.set('view engine','jade');

app.get('/',function(req,res)

{

});

var server=app.listen(3000,function()

{

});

1. In the first statement itself, we are using the "require" keyword to include the express module. The "express" module is an optimized JavaScript library for Node.js development. This is one of the most commonly used Node.js modules.
2. After the module is included, in order to use the functionality within the module, an object needs to be created. Here an object of the express module is created.
3. Once the module is included using the "require" command and an "object" is created, the required methods of the express module can be invoked. Here we are using the set command to set the view engine, which is used to set the templating engine used in Node.js.

**Note:-**(Just for the reader's understanding; a templating engine is an approach for injecting values in an application by picking up data from data files. This concept is pretty famous in Angular JS wherein the curly braces {{ key }} is used to substitutes values in the web page. The word 'key' in the curly braces basically denotes the variable which will be substituted by a value when the page is displayed.)

1. Here we are using the listen to method to make the application listen on a particular port number.

**Creating NPM modules**

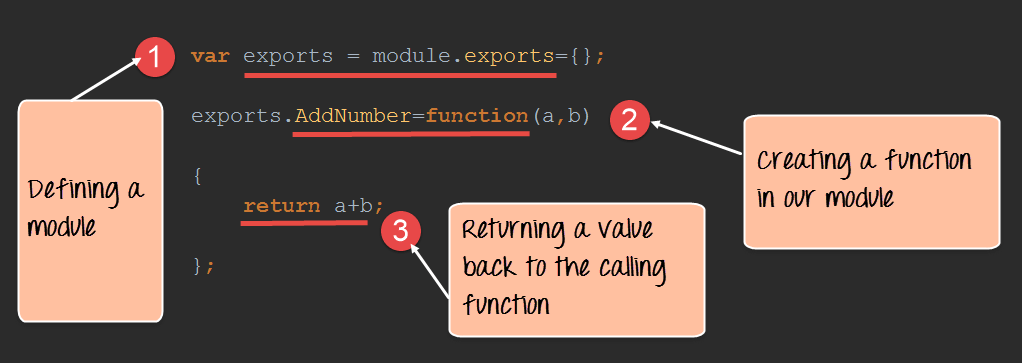
Node.js has the ability to create custom modules and allows you to include those custom modules in your Node.js application.

Let's look at a simple example of how we can create our own module and include that module in our main application file. Our module will just do a simple task of adding two numbers.

Let's follow the below steps to see how we can create modules and include them in our application.

**Step 1)** Create a file called "Addition.js" and include the below code. This file will contain the logic for your module.

Below is the code which would go into this file;

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul2.png)

var exports=module.exports={};

exports.AddNumber=function(a,b)

{

return a+b;

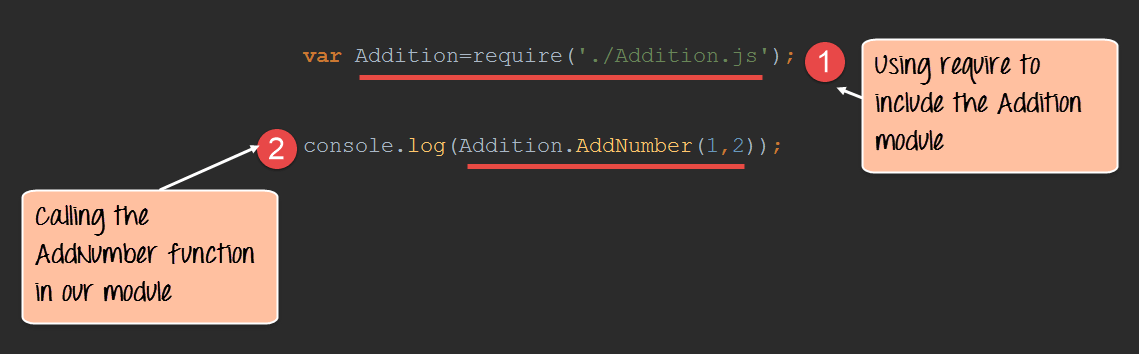
};

1. The "exports" keyword is used to ensure that the functionality defined in this file can actually be accessed by other files.
2. We are then defining a function called 'AddNumber'. This function is defined to take 2 parameters, a and b. The function is added to the module "exports" to make the function as a public function that can be accessed by other application modules.
3. We are finally making our function return the added value of the parameters.

Now that we have created our custom module which has the functionality of adding 2 numbers. It's now time to create an application, which will call this module.

In the next step, we will actually see how to create the application which will call our custom module.

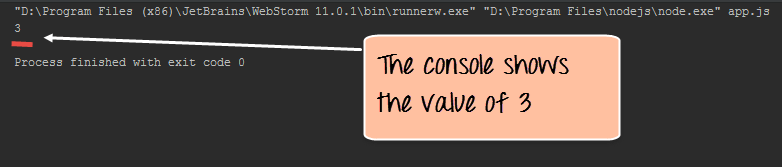
**Step 2)** Create a file called "app.js," which is your main application file and add the below code

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul3.png)

var Addition=require('./Addition.js');

console.log(Addition.AddNumber(1,2));

1. We are using the "require" keyword to include the functionality in the Addition.js file.
2. Since the functions in the Addition.js file are now accessible, we can now make a call to the AddNumber function. In the function, we are passing 2 numbers as parameters. We are then displaying the value in the console.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul4.png)

**Output**:

* When you run the app.js file, you will get an output of value 3 in the console log.
* The result is because the AddNumber function in the Addition.js file was called successfully, and the returned value of 3 was displayed in the console.

**Note:** - We are not using the "Node package manager" as of yet to install our Addition.js module. This is because the module is already part of our project on the local machine. The Node package manager comes in the picture when you publish a module on the internet, which we see in the subsequent topic.

**Extending modules**

When creating modules, it is also possible to extend or inherit one module from another.

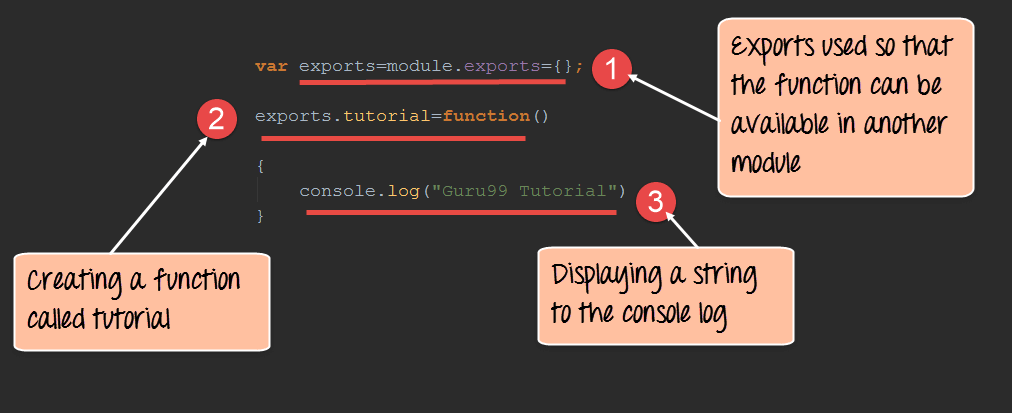
In modern-day programming, it's quite common to build a library of common modules and then extend the functionality of these common modules if required.

Let's look at an example of how we can extend modules in Node.js.

**Step 1)** Create the base module.

In our example, create a file called "Tutorial.js" and place the below code.

In this code, we are just creating a function which returns a string to the console. The string returned is "- Tutorial".

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul5.png)

var exports=module.exports={};

exports.tutorial=function()

{

console.log("- Tutorial")

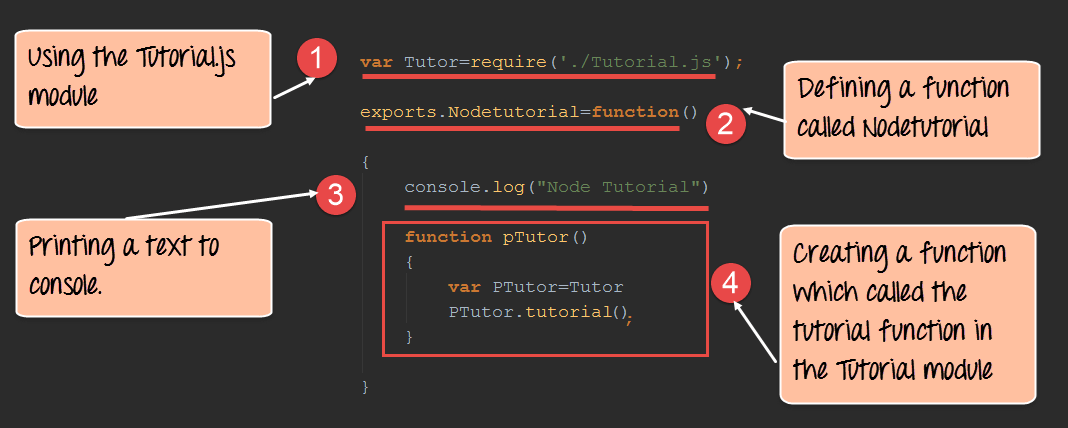
}

1. The exports module is used so that whatever function is defined in this file can be available in other modules in Node.js
2. We are creating a function called tutorial which can be used in other Node.js modules.
3. We are displaying a string "- Tutorial" in the console when this function is called.

Now that we have created our base module called Tutorial.js. It's now time to create another module which will extend this base module.

We will explore how to do this in the next step.

**Step 2)** Next, we will create our extended module. Create a new file called "NodeTutorial.js" and place the below code in the file.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul6.png)

var Tutor=require('./Tutorial.js');

exports.NodeTutorial=function()

{

console.log("Node Tutorial")

function pTutor()

{

var PTutor=Tutor

PTutor.tutorial();

}

}

Or

var Tutor=require('./Tutorial.js');

exports.NodeTutorial=function()

{

console.log("Node Tutorial")

this.pTutor = function ()

{

var PTutor=Tutor

PTutor.tutorial();

}

}

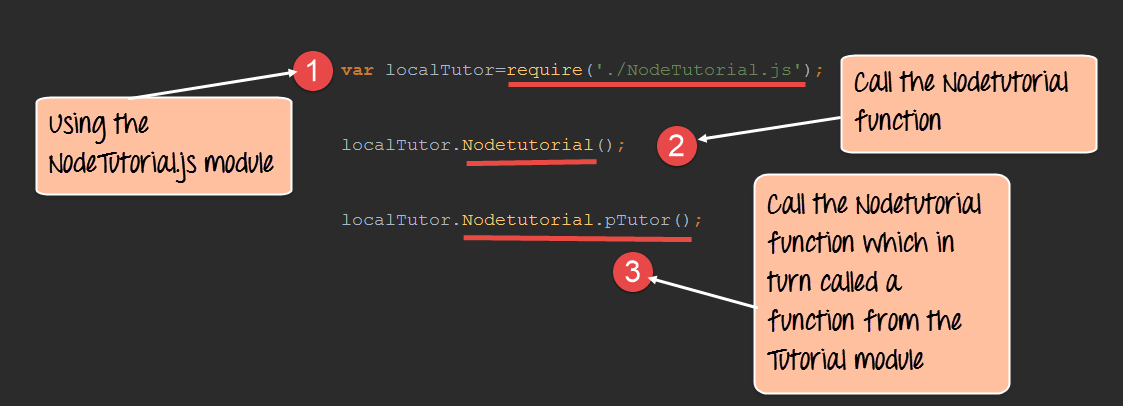
Note, the following key points about the above code

1. We are using the "require" function in the new module file itself. Since we are going to extend the existing module file "Tutorial.js", we need to first include it before extending it.
2. We then create a function called "Nodetutorial." This function will do 2 things,

* It will send a string "Node Tutorial" to the console.
* It will send the string "- Tutorial" from the base module "Tutorial.js" to our extended module "NodeTutorial.js".

1. Here we are carrying out the first step to send a string to "Node Tutorial" to the console.
2. The next step is to call the function from our Tutorial module, which will output the string "- Tutorial" to the console.log.

**Step 3)** Create your main app.js file, which is your main application file and include the below code.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul7.png)

var localTutor=require('./NodeTutorial.js');

localTutor.NodeTutorial();

localTutor.NodeTutorial.pTutor();

Or use this code

var tut = new localTutor.NodeTutorial(); // Create and save object

tut.pTutor(); // Call function on object

The above code does the following things;

1. Our main application file now calls the "NodeTutorial" module.
2. We are calling the "NodeTutorial" function. By calling this function, the text "Node Tutorial" will be displayed in the console log.
3. Since we have extended our Tutorial.js module and exposed a function called pTutor. It also calls the tutorial module in the Tutorial.js module, and the text "- Tutorial" will be displayed to the console as well.

**Output:**

Since we have executed the above app.js code using Node, we will get the following output in the console.log file

* Node Tutorial
* - Tutorial

**Publishing NPM(Node Package Manager) Modules**

One can publish their own module to their own Github repository.

By publishing your module to a central location, you are then not burdened with having to install yourself on every machine that requires it.

Instead, you can use the install command of npm and install your published npm module.

The following steps need to be followed to publish your npm module

**Step 1)** Create your repository on GitHub (an online code repository management tool). It can be used for hosting your code repositories.

**Step 2**) You need to tell your local npm installation on who you are. Which means that we need to tell npm who is the author of this module, what is the email id and any company URL, which is available which needs to be associated with this id. All of these details will be added to your npm module when it is published.

The below commands sets the name, email and URL of the author of the npm module.

npm set init.author.name "-."

npm set init.author.email "[-@gmail.com](mailto:guru99@gmail.com) "

npm set init.author.url [http://-.com](https://www.guru99.com/)

**Step 3)** The next step is to login into npm using the credentials provided in the last step. To login, you need to use the below command

npm login

**Step 4)** Initialize your package – The next step is to initialize the package to create the package.json file. This can be done by issuing the below command

npm init

When you issue the above command, you will be prompted for some questions. The most important one is the version number for your module.

**Step 5)** Publish to GitHub – The next step is to publish your source files to GitHub. This can be done by running the below commands.

git add.

git commit -m "Initial release"

git tag v0.0.1

git push origin master --tags

**Step 6)** Publish your module – The final bit is to publish your module into the npm registry. This is done via the below command.

npm publish

**Managing third party packages with npm**

As we have seen, the "Node package manager" has the ability to manage modules, which are required by Node.js applications.

Let's look at some of the functions available in the node package manager for managing modules

1. Installing packages in global mode – Modules can be installed at the global level, which just basically means that these modules would be available for all Node.js projects on a local machine. The example below shows how to install the "express module" with the global option.

**npm install express –global**

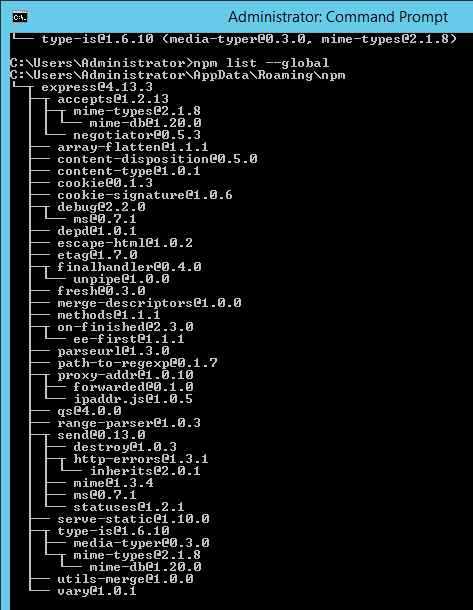
The global option in the above statement is what allows the modules to be installed at a global level.

1. Listing all of the global packages installed on a local machine. This can be done by executing the below command in the command prompt

**npm list --global**

Below is the output which will be shown, if you have previously installed the "express module" on your system.

Here you can see the different modules installed on the local machine.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul8.png)

1. Installing a specific version of a package – Sometimes there may be a requirement to install just the specific version of a package. Once you know package name and the relevant version that needs to be installed, you can use the npm install command to install that specific version.

The example below shows how to install the module called underscore with a specific version of 1.7.0

npm install **underscore@1.7.0**

1. Updating a package version – Sometimes you may have an older version of a package in a system, and you may want to update to the latest one available in the market. To do this, one can use the npm update command. The example below shows how to update the underscore package to the latest version

**npm update underscore**

1. Searching for a particular package – To search whether a particular version is available on the local system or not, you can use the search command of npm. The example below will check if the express module is installed on the local machine or not.

**npm search express**

1. Uninstalling a package – The same in which you can install a package, you can also uninstall a package. The uninstallation of a package is done with the uninstallation command of npm. The example below shows how to uninstall the express module

**npm uninstall express**

**What is the package.json file**

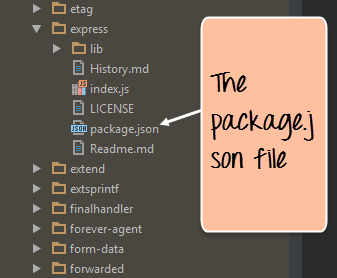
The "package.json" file is used to hold the **metadata about a particular project**. This information provides the Node package manager the necessary information to understand how the project should be handled along with its dependencies.

The package.json files contain information such as the project description, the version of the project in a particular distribution, license information, and configuration data.

The package.json file is normally located at the root directory of a Node.js project.

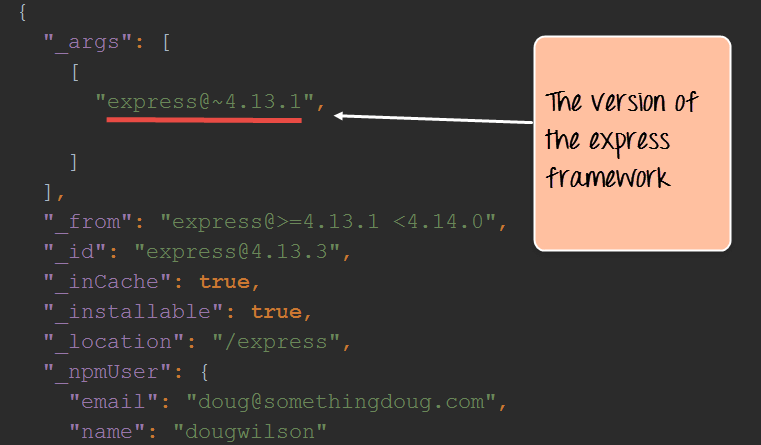
Let's take an example of how the structure of a module looks when it is installed via npm.

The below snapshot shows the file contents of the express module when it is included in your Node.js project. From the snapshot, you can see the package.json file in the express folder.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul9.png)

If you open the package.json file, you will see a lot of information in the file.

Below is a snapshot of a portion of the file. The **express@~4.13.1** mentions the version number of the express module being used.

[](https://www.guru99.com/images/NodeJS/010716_0523_NodejsModul10.png)

**Summary**

* A module in Node.js is a logical encapsulation of code in a single unit. Separation into modules makes code more manageable and maintainable for future purposes
* There are many modules available in the market which can be used within Node.js such as express, underscore, MongoDB, etc.
* The node package manager (npm) is used to download and install modules which can then be used in a Node.js application.
* One can create custom NPM modules, extend these modules, and also publish these modules.
* The Node package manager has a complete set of commands to manage the npm modules on the local system such as the installation, un-installation, searching, etc.
* The package.json file is used to hold the entire metadata information for an npm module.

**Create HTTP Web Server in Node.js: Complete Tutorial**

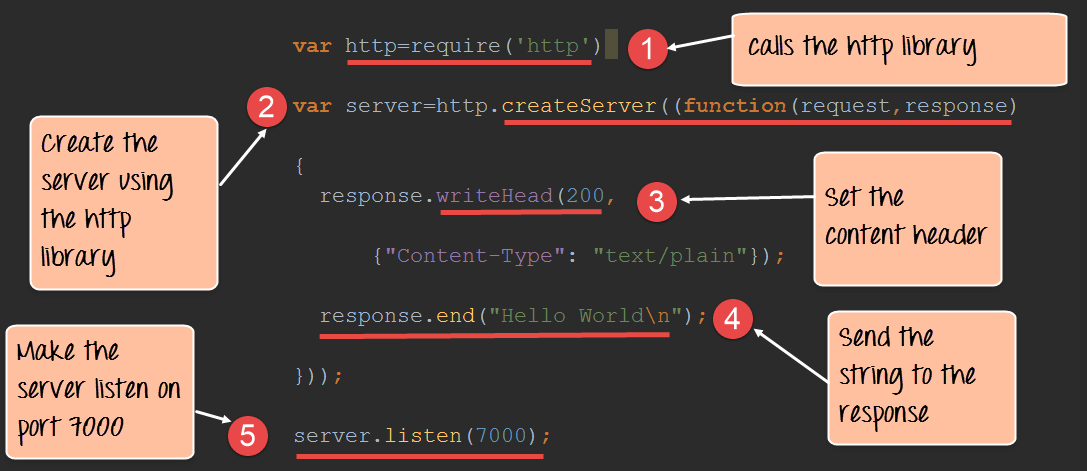
The Node.js framework is mostly used to create server-based applications. The framework can easily be used to create web servers which can serve content to users.

There are a variety of modules such as the "http" and "request" module, which helps in processing server related requests in the webserver space. We will have a look at how we can create a basic web server application using Node js.

**Node as a web server using HTTP**

Let's look at an example of how to create and run our first Node js application.

Our application is going to create a simple server module which will listen on port no 7000. If a request is made through the browser on this port no, then server application will send a 'Hello World' response to the client.

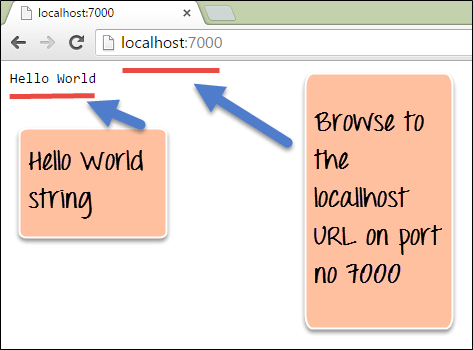
[](https://www.guru99.com/images/NodeJS/010716_0600_NodejsHttpT1.png)

**Code Explanation:**

1. The basic functionality of the require function is that it reads a[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)file, executes the file, and then proceeds to return the exports object. So in our case, since we want to use the functionality of the http module, we use the require function to get the desired functions from the http module so that it can be used in our application.
2. In this line of code, we are creating a server application which is based on a simple function. This function is called whenever a request is made to our server application.
3. When a request is received, we are saying to send a response with a header type of '200.' This number is the normal response which is sent in an http header when a successful response is sent to the client.
4. In the response itself, we are sending the string 'Hello World.'
5. We are then using the server.listen function to make our server application listen to client requests on port no 7000. You can specify any available port over here.

If the command is executed successfully, the following Output will be shown when you run your code in the browser.

**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0600_NodejsHttpT2.png)

From the output,

* You can clearly see that if we browse to the URL of localhost on port 7000, you will see the string 'Hello World' displayed in the page.
* Because in our code we have mentioned specifically for the server to listen on port no 7000, we are able to view the output when browsing to this url.

Here is the code for your reference

var http=require('http')

var server=http.createServer((function(request,response)

{

response.writeHead(200,

{"Content-Type" : "text/plain"});

response.end("Hello World\n");

}));

server.listen(7000);

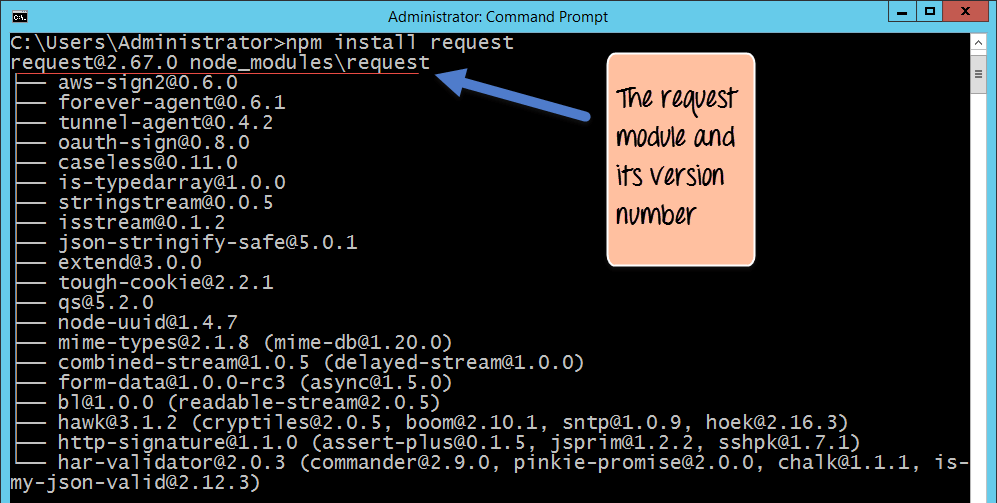
**Handling GET Requests in Node.js**

Making a GET Request to get the data from another site is relatively very simple in Node.js. To make a Get request in the node, we need to first have the request module installed. This can be done by executing the following line in the command line

**npm install request**

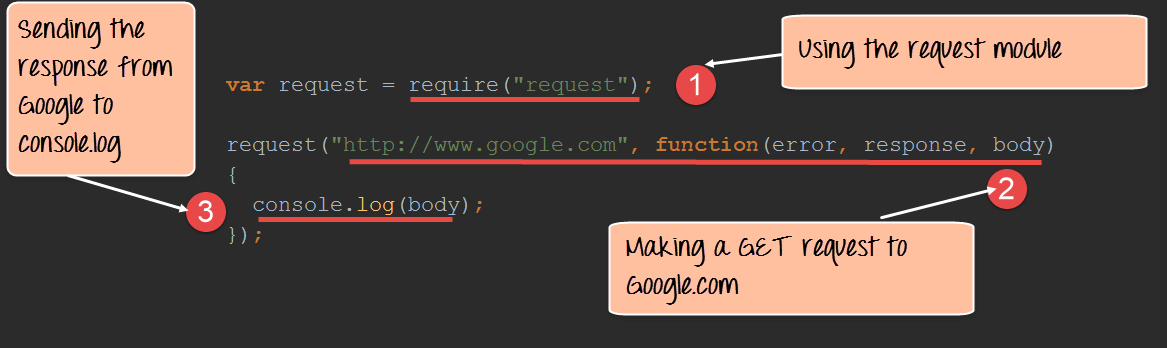
The above command requests the Node package manager to download the required request modules and install them accordingly.

When your npm module has been installed successfully, the command line will show the installed module name and version: <name>@<version>.

[](https://www.guru99.com/images/NodeJS/010716_0600_NodejsHttpT3.png)

In the above snapshot, you can see that the 'request' module along with the version number 2.67.0 was downloaded and installed.

Now let's see the code which can make use of this 'request' command.

[](https://www.guru99.com/images/NodeJS/010716_0600_NodejsHttpT4.png)

**Code Explanation:**

1. We are using the 'require' module which was installed in the last step. This module has the necessary functions which can be used to make GET requests to websites.
2. We are making a GET Request to www.google.com and subsequently calling a function when a response is received. When a response is received the parameters(error, response, and body) will have the following values
   1. Error – In case there is any error received when using the GET request, it will be recorded here.
   2. Response- The response will have the http headers which are sent back in the response.
   3. Body- The body will contain the entire content of the response sent by Google.
3. In this, we are just writing the content received in the body parameter to the console.log file. So basically, whatever we get by going to **www.google.com** will be written to the console.log.

Here is the code for your reference

var request = require("request");

request("http://www.google.com",function(error,response,body)

{

console.log(body);

});

**Summary**

* The Node.js framework can be used to develop web servers using the 'http' module. The application can be made to listen on a particular port and send a response to the client whenever a request is made to the application.
* The 'request' module can be used to get information from web sites. The information would contain the entire content of the web page requested from the relevant web site.

**Node.js Express FrameWork Tutorial - Learn in 10 Minutes**

In this tutorial, we will study the Express framework. This framework is built in such a way that it acts as a minimal and flexible Node.js web application framework, providing a robust set of features for building single and multipage, and hybrid web application.

In this tutorial, you will learn-

* [What is Express.js?](https://www.guru99.com/node-js-express.html#1)
* [Installing and using Express](https://www.guru99.com/node-js-express.html#2)
* [What are Routes?](https://www.guru99.com/node-js-express.html#3)
* [Sample Web server using express.js](https://www.guru99.com/node-js-express.html#4)

**What is Express.js?**

Express.js is a Node js web application server framework, which is specifically designed for building single-page, multi-page, and hybrid web applications.

It has become the standard server framework for node.js. Express is the backend part of something known as the MEAN stack.

The MEAN is a free and open-source[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)software stack for building dynamic web sites and web applications which has the following components;

**1) MongoDB** - The standard NoSQL database

**2) Express.js** - The default web applications framework

**3) Angular.js** - The JavaScript MVC framework used for web applications

**4) Node.js** - Framework used for scalable server-side and networking applications.

The Express.js framework makes it very easy to develop an application which can be used to handle multiple types of requests like the GET, PUT, and POST and DELETE requests.

**Installing and using Express**

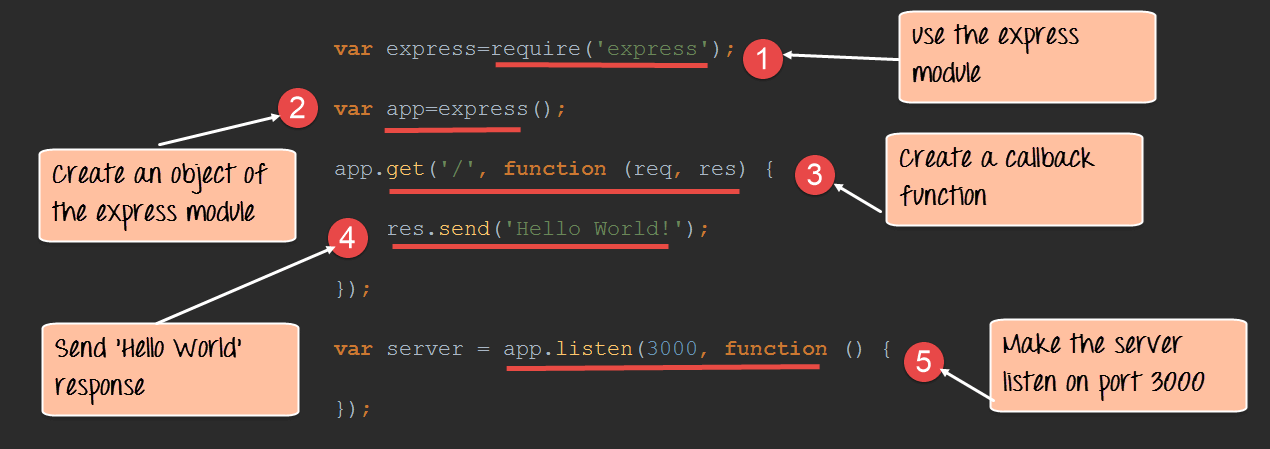
Express gets installed via the Node Package Manager. This can be done by executing the following line in the command line

**npm install express**

The above command requests the Node package manager to download the required express modules and install them accordingly.

Let's use our newly installed Express framework and create a simple "Hello World" application.

Our application is going to create a simple server module which will listen on port number 3000. In our example, if a request is made through the browser on this port number, then server application will send a 'Hello' World' response to the client.

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre1.png)

var express=require('express');

var app=express();

app.get('/',function(req,res)

{

res.send('Hello World!');

});

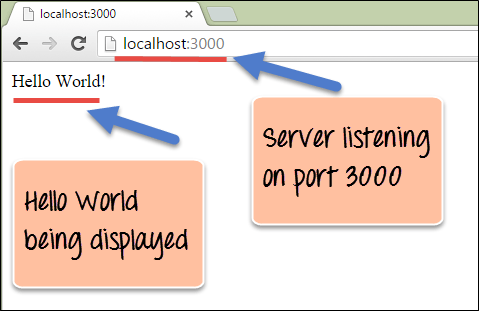
var server=app.listen(3000,function() {});

**Code Explanation:**

1. In our first line of code, we are using the require function to include the "express module."
2. Before we can start using the express module, we need to make an object of it.
3. Here we are creating a callback function. This function will be called whenever anybody browses to the root of our web application which is **http://localhost:3000** . The callback function will be used to send the string 'Hello World' to the web page.
4. In the callback function, we are sending the string "Hello World" back to the client. The 'res' parameter is used to send content back to the web page. This 'res' parameter is something that is provided by the 'request' module to enable one to send content back to the web page.
5. We are then using the listen to function to make our server application listen to client requests on port no 3000. You can specify any available port over here.

If the command is executed successfully, the following Output will be shown when you run your code in the browser.

**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre2.png)

From the output,

* You can clearly see that we if browse to the URL of localhost on port 3000, you will see the string 'Hello World' displayed on the page.
* Because in our code we have mentioned specifically for the server to listen on port no 3000, we are able to view the output when browsing to this URL.

**What are Routes?**

Routing determine the way in which an application responds to a client request to a particular endpoint.

For example, a client can make a GET, POST, PUT or DELETE http request for various URL such as the ones shown below;

http://localhost:3000/Books

http://localhost:3000/Students

In the above example,

* If a GET request is made for the first URL, then the response should ideally be a list of books.
* If the GET request is made for the second URL, then the response should ideally be a list of Students.
* So based on the URL which is accessed, a different functionality on the webserver will be invoked, and accordingly, the response will be sent to the client. This is the concept of routing.

Each route can have one or more handler functions, which are executed when the route is matched.

The general syntax for a route is shown below

app.METHOD(PATH, HANDLER)

Wherein,

1) app is an instance of the express module

2) METHOD is an HTTP request method (GET, POST, PUT or DELETE)

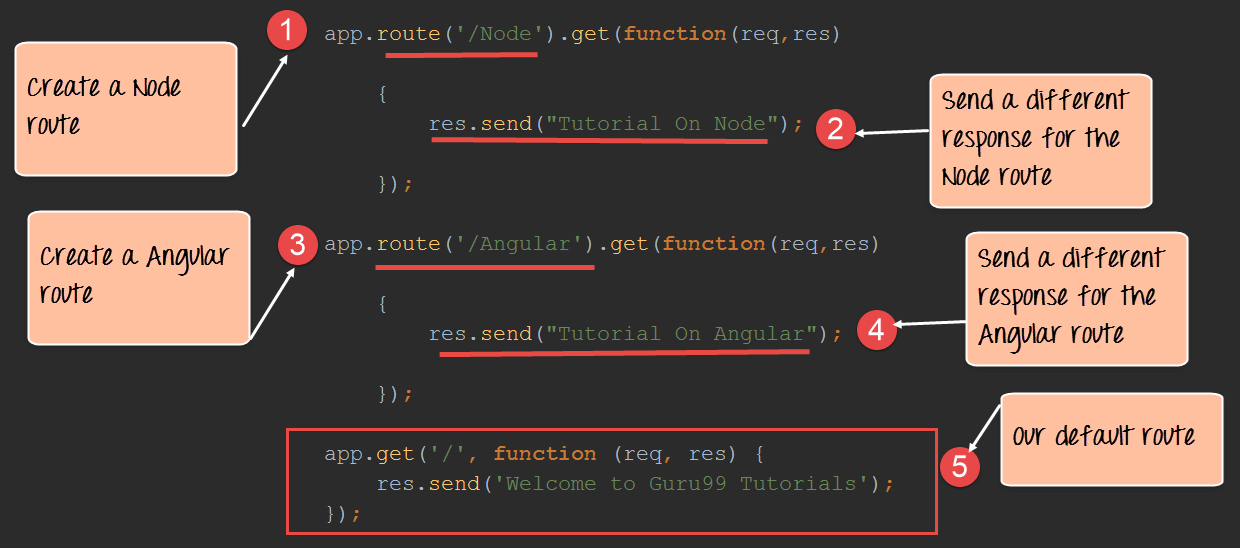
3) PATH is a path on the server.

4) HANDLER is the function executed when the route is matched.

Let's look at an example of how we can implement routes in the express. Our example will create 3 routes as

1. A /Node route which will display the string "Tutorial on Node" if this route is accessed
2. A /Angular route which will display the string "Tutorial on Angular" if this route is accessed
3. A default route / which will display the string "Welcome to - Tutorials."

Our basic code will remain the same as previous examples. The below snippet is an add-on to showcase how routing is implemented.

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre3.png)

var express = require('express');

var app = express();

app.route('/Node',get(function(req,res)

{

res.send("Tutorial on Node");

});

app.route('/Angular',get(function(req,res)

{

res.send("Tutorial on Angular");

});

app.get('/',(function(req,res){

res.send('Welcome to - Tutorials');

}));

**Code Explanation:**

1. Here we are defining a route if the URL **http://localhost:3000/Node** is selected in the browser. To the route, we are attaching a callback function which will be called when we browse to the Node URL.

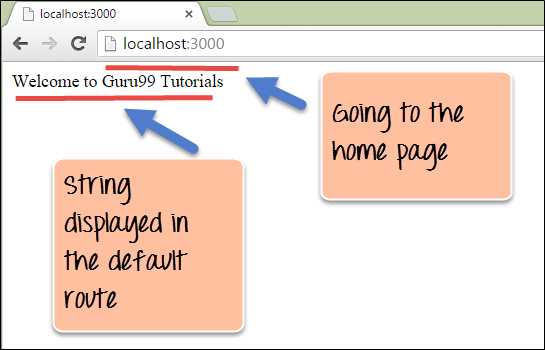
The function has 2 parameters.

* The main parameter we will be using is the 'res' parameter, which can be used to send information back to the client.
* The 'req' parameter has information about the request being made. Sometimes additional parameters could be sent as part of the request being made, and hence the 'req' parameter can be used to find the additional parameters being sent.

1. We are using the send function to send the string "Tutorial on Node" back to the client if the Node route is chosen.
2. Here we are defining a route if the URL **http://localhost:3000/Angular** is selected in the browser. To the route, we are attaching a callback function which will be called when we browse to the Angular URL.
3. We are using the send function to send the string "Tutorial on Angular" back to the client if the Angular route is chosen.
4. This is the default route which is chosen when one browses to the route of the application – **http://localhost:3000**. When the default route is chosen, the message "Welcome to - Tutorials" will be sent to the client.

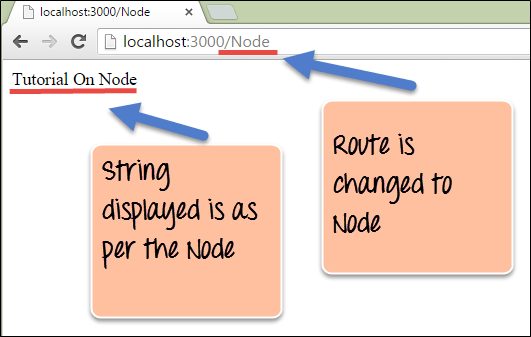
If the command is executed successfully, the following Output will be shown when you run your code in the browser.

**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre4.png)

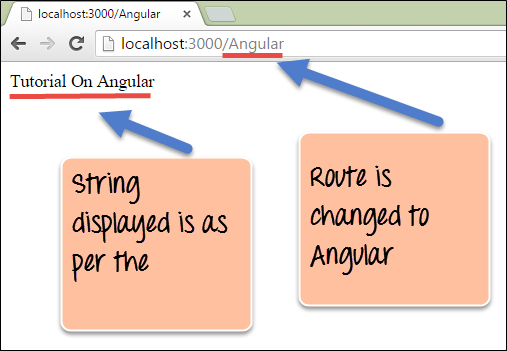
From the output,

* You can clearly see that we if browse to the URL of localhost on port 3000, you will see the string 'Welcome to - Tutorials' displayed on the page.
* Because in our code, we have mentioned that our default URL would display this message.

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre5.png)

From the output,

* You can see that if the URL has been changed to /Node, the respective Node route would be chosen and the string "Tutorial On Node' is displayed.

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre6.png)

From the output,

* You can see that if the URL has been changed to /Angular, the respective Node route would be chosen and the string "Tutorial On Angular" is displayed.

**Sample Web server using express.js**

From our above example, we have seen how we can decide on what output to show based on routing. This sort of routing is what is used in most modern-day web applications. The other part of a web server is about using templates in Node js.

When creating quick on-the-fly Node applications, an easy and fast way is to use templates for the application. There are many frameworks available in the market for making templates. In our case, we will take the example of the jade framework for templating.

Jade gets installed via the Node Package manager. This can be done by executing the following line in the command line

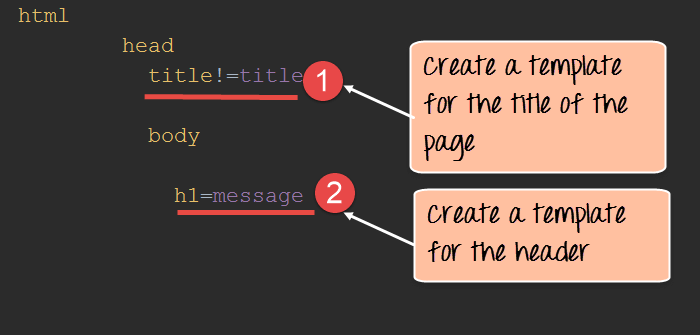
**npm install jade**

The above command requests the Node package manager to download the required jade modules and install them accordingly.

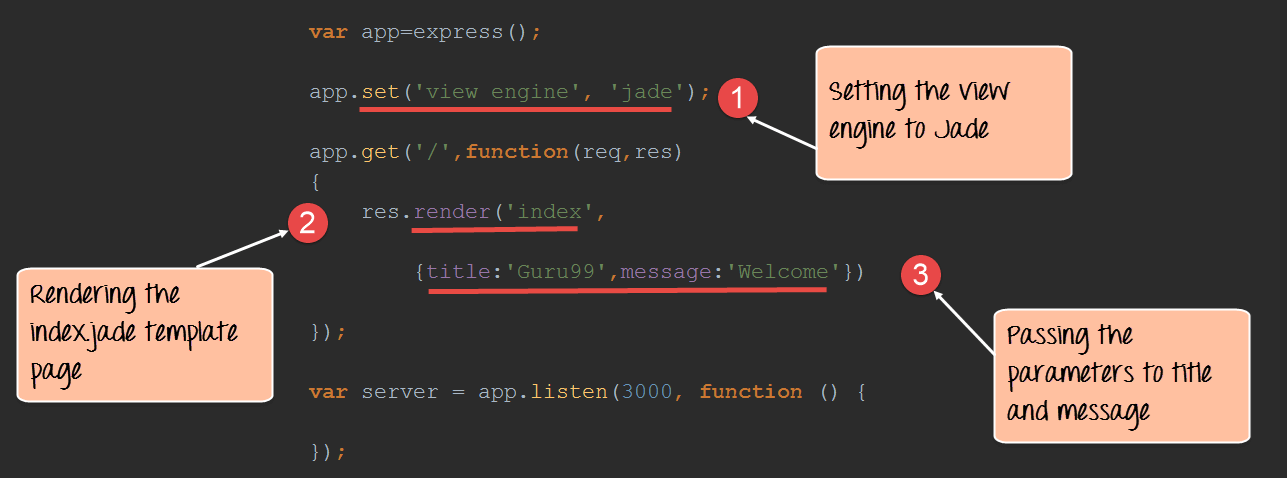
**NOTE:** In the latest version of Node jade has been deprecated. Instead, use pug.

Let's use our newly installed jade framework and create some basic templates.

**Step 1)** The first step is to create a jade template. Create a file called index.jade and insert the below code. Ensure to create the file in "views" folder

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre7.png)

1. Here we are specifying that the title of the page will be changed to whatever value is passed when this template gets invoked.
2. We are also specifying that the text in the header tag will get replaced to whatever gets passed in the jade template.

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre8.png)

var express=require('express');

var app=express();

app.set('view engine','jade');

app.get('/',function(req,res)

{

res.render('index',

{title:'-',message:'Welcome'})

});

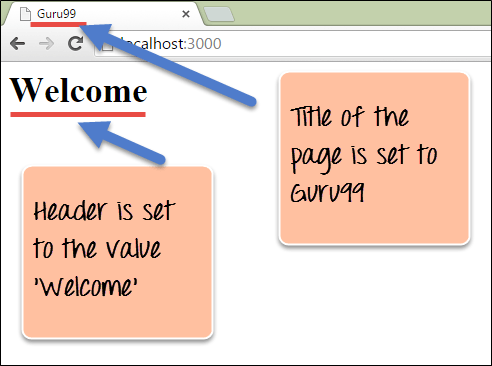
var server=app.listen(3000,function() {});

**Code Explanation:**

1. The first thing to specify in the application is "view engine" that will be used to render the templates. Since we are going to use jade to render our templates, we specify this accordingly.
2. The render function is used to render a web page. In our example, we are rendering the template (index.jade) which was created earlier.
3. We are passing the values of - and "Welcome" to the parameters "title" and "message" respectively. These values will be replaced by the 'title', and 'message' parameters declared in the index.jade template.

If the command is executed successfully, the following Output will be shown when you run your code in the browser.

**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0613_NodejsExpre9.png)

From the output,

* We can see that the title of the page gets set to - and the header of the page gets set to "Welcome."
* This is because of the jade template which gets invoked in our node js application.

**Summary**

* The express framework is the most common framework used for developing Node js applications. The express framework is built on top of the node.js framework and helps in fast-tracking development of server-based applications.
* Routes are used to divert users to different parts of the web applications based on the request made. The response for each route can be varied depending on what needs to be shown to the user.
* Templates can be used to inject content in an efficient manner. Jade is one of the most popular templating engines used in Node.js applications.

**Node.js MongoDB Tutorial with Examples**

Mostly all modern-day web applications have some sort of data storage system at the backend. For example, if you take the case of a web shopping application, data such as the price of an item would be stored in the database.

The Node js framework can work with databases with both relational (such as Oracle and MS SQL Server) and non-relational databases (such as MongoDB). In this tutorial, we will see how we can use databases from within Node js applications.

In this tutorial, you will learn-

* [Node.js and NoSQL Databases](https://www.guru99.com/node-js-mongodb.html#1)
* [Using MongoDB and Node.js](https://www.guru99.com/node-js-mongodb.html#2)
* [How to build a node express app with MongoDB to store and serve content](https://www.guru99.com/node-js-mongodb.html#3)

**Node.js and NoSQL Databases**

Over the years, NoSQL database such as[MongoDB](https://www.guru99.com/mongodb-tutorials.html)and MySQL have become quite popular as databases for storing data. The ability of these databases to store any type of content and particularly in any type of format is what makes these databases so famous.

Node.js has the ability to work with both MySQL and MongoDB as databases. In order to use either of these databases, you need to download and use the required modules using the Node package manager.

For MySQL, the required module is called "mysql" and for using MongoDB the required module to be installed is "Mongoose."

With these modules, you can perform the following operations in Node.js

1. Manage the connection pooling – Here is where you can specify the number of MySQL database connections that should be maintained and saved by Node.js.
2. Create and close a connection to a database. In either case, you can provide a callback function which can be called whenever the "create" and "close" connection methods are executed.
3. Queries can be executed to get data from respective databases to retrieve data.
4. Data manipulation, such as inserting data, deleting, and updating data can also be achieved with these modules.

For the remaining topics, we will look at how we can work with MongoDB databases within Node.js.

**Using MongoDB and Node.js**

As discussed in the earlier topic, MongoDB is one of the most popular databases used along with Node.js.

During this chapter, we will see

How we can establish connections with a MongoDB database

How we can perform the normal operations of reading data from a database as well as inserting, deleting, and updating records in a MongoDB database.

For the purpose of this chapter, let's assume that we have the below MongoDB data in place.

Database name: EmployeeDB

Collection name: Employee

Documents

{

{Employeeid : 1, Employee Name : -},

{Employeeid : 2, Employee Name : Joe},

{Employeeid : 3, Employee Name : Martin},

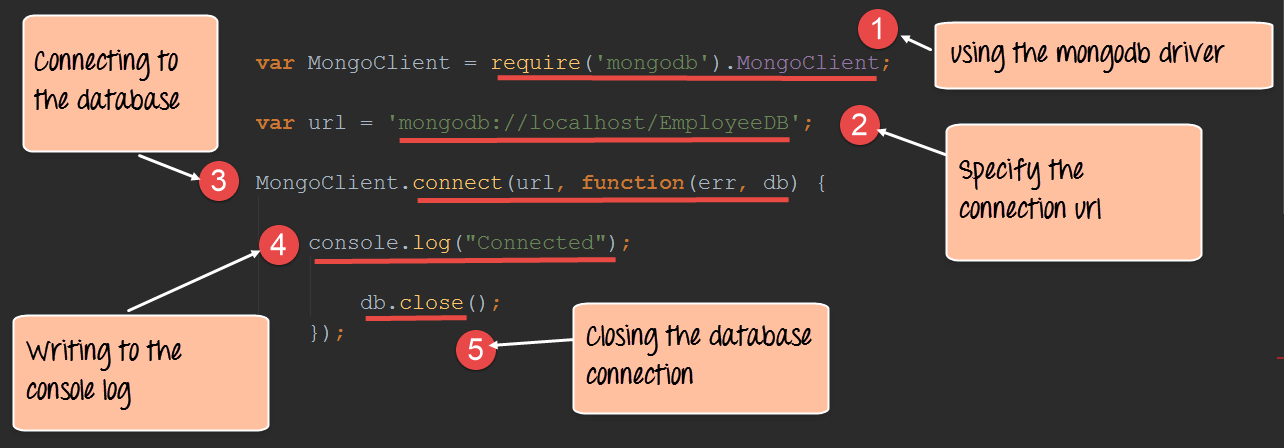
}

1. **Installing the NPM Modules**

You need a driver to access Mongo from within a Node application. There are a number of Mongo drivers available, but MongoDB is among the most popular. To install the MongoDB module, run the below command

**npm install mongodb**

1. **Creating and closing a connection to a MongoDB database.**The below code snippet shows how to create and close a connection to a MongoDB database.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo1.png)

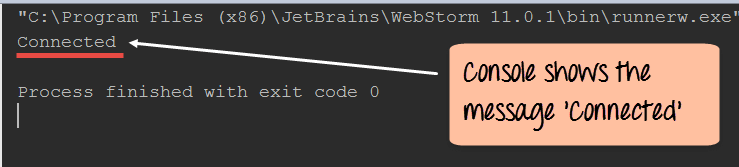
**Code Explanation:**

1. The first step is to include the mongoose module, which is done through the require function. Once this module is in place, we can use the necessary functions available in this module to create connections to the database.
2. Next, we specify our connection string to the database. In the connect string, there are 3 key values which are passed.

* The first is 'mongodb' which specifies that we are connecting to a mongoDB database.
* The next is 'localhost' which means we are connecting to a database on the local machine.
* The next is 'EmployeeDB' which is the name of the database defined in our MongoDB database.

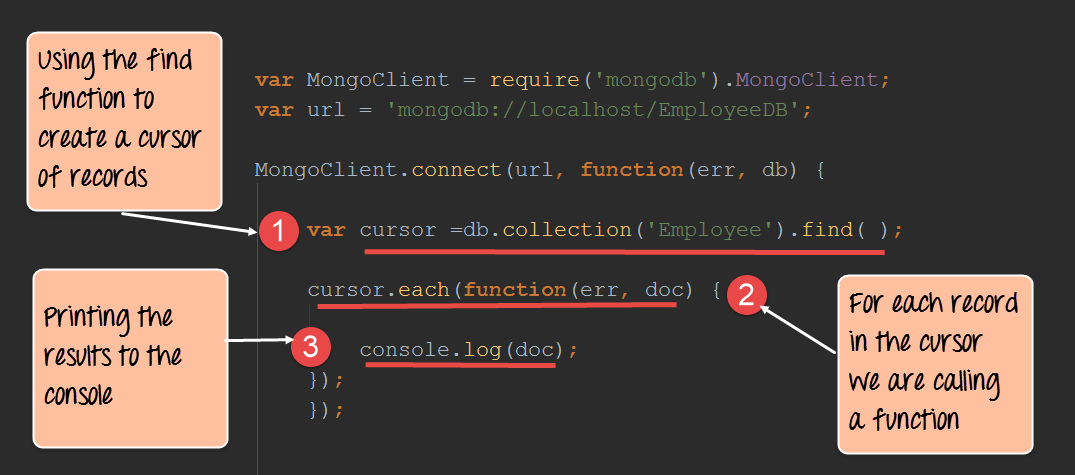
1. The next step is to actually connect to our database. The connect function takes in our URL and has the facility to specify a callback function. It will be called when the connection is opened to the database. This gives us the opportunity to know if the database connection was successful or not.
2. In the function, we are writing the string "Connection established" to the console to indicate that a successful connection was created.
3. Finally, we are closing the connection using the db.close statement.

If the above code is executed properly, the string "Connected" will be written to the console as shown below.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo2.png)

1. **Querying for data in a MongoDB database** – Using the MongoDB driver we can also fetch data from the MongoDB database.

The below section will show how we can use the driver to fetch all of the documents from our Employee collection in our EmployeeDB database. This is the collection in our MongoDB database, which contains all the employee-related documents. Each document has an object id, Employee name, and employee id to define the values of the document.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo3.png)

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

var cursor = db.collection('Employee').find();

cursor.each(function(err, doc) {

console.log(doc);

});

});

**Code Explanation:**

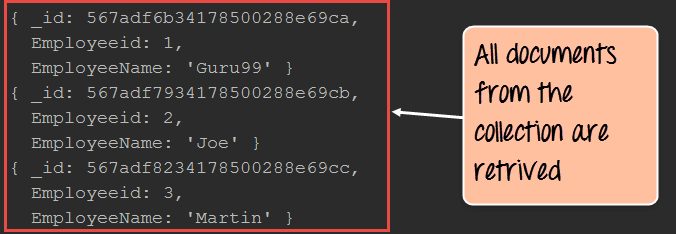
1. In the first step, we are creating a cursor (A cursor is a pointer which is used to point to the various records fetched from a database. The cursor is then used to iterate through the different records in the database. Here we are defining a variable name called cursor which will be used to store the pointer to the records fetched from the database. ) which points to the records which are fetched from the MongoDb collection. We also have the facility of specifying the collection 'Employee' from which to fetch the records. The find() function is used to specify that we want to retrieve all of the documents from the MongoDB collection.
2. We are now iterating through our cursor and for each document in the cursor we are going to execute a function.
3. Our function is simply going to print the contents of each document to the console.

**Note: -** It is also possible to fetch a particular record from a database. This can be done by specifying the search condition in the find() function. For example, suppose if you just wanted to fetch the record which has the employee name as -, then this statement can be written as follows

var cursor=db.collection('Employee').find({EmployeeName: -})

If the above code is executed successfully, the following output will be displayed in your console.

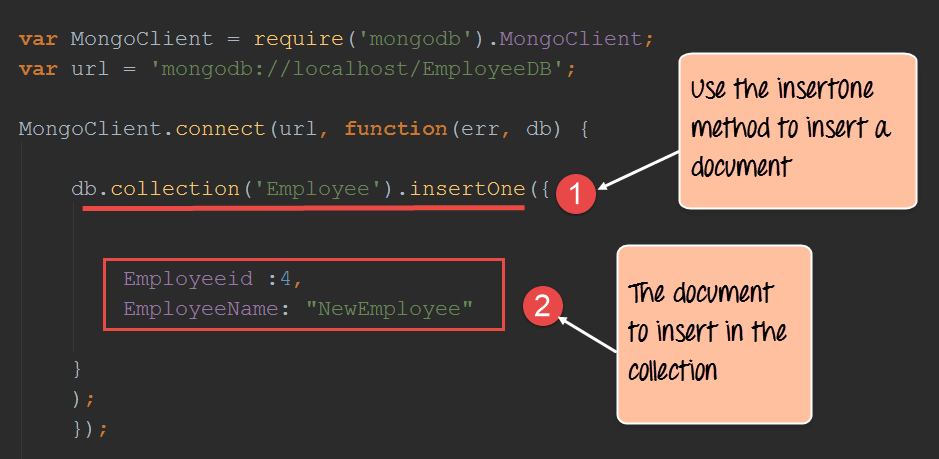
**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo4.png)

From the output,

* You will be able to clearly see that all the documents from the collection are retrieved. This is possible by using the find() method of the mongoDB connection (db) and iterating through all of the documents using the cursor.

1. **Inserting documents in a collection** – Documents can be inserted into a collection using the insertOne method provided by the MongoDB library. The below code snippet shows how we can insert a document into a mongoDB collection.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo5.png)

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

db.collection('Employee').insertOne({

Employeeid: 4,

EmployeeName: "NewEmployee"

});

});

**Code Explanation:**

1. Here we are using the insertOne method from the MongoDB library to insert a document into the Employee collection.
2. We are specifying the document details of what needs to be inserted into the Employee collection.

If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "NewEmployee" inserted into the Employee collection.

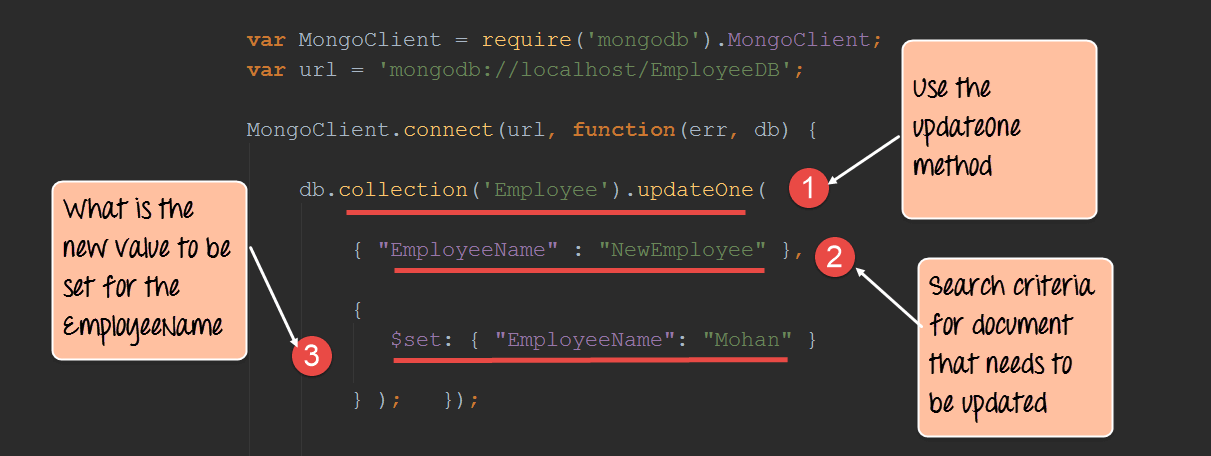
**Note:** The console will not show any output because the record is being inserted in the database and no output can be shown here.

To check that the data has been properly inserted in the database, you need to execute the following commands in MongoDB

1. Use EmployeeDB
2. db.Employee.find({Employeeid :4 })

The first statement ensures that you are connected to the EmployeeDb database. The second statement searches for the record which has the employee id of 4.

1. **Updating documents in a collection** - Documents can be updated in a collection using the updateOne method provided by the MongoDB library. The below code snippet shows how to update a document in a mongoDB collection.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo6.png)

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

db.collection('Employee').updateOne({

"EmployeeName": "NewEmployee"

}, {

$set: {

"EmployeeName": "Mohan"

}

});

});

**Code Explanation:**

1. Here we are using the "updateOne" method from the MongoDB library, which is used to update a document in a mongoDB collection.
2. We are specifying the search criteria of which document needs to be updated. In our case, we want to find the document which has the EmployeeName of "NewEmployee."
3. We then want to set the value of the EmployeeName of the document from "NewEmployee" to "Mohan".

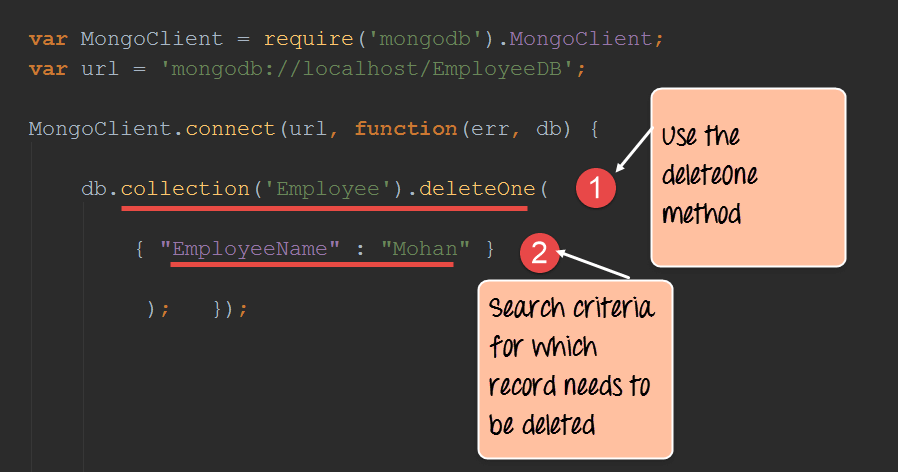
If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "Mohan" updated in the Employee collection.

To check that the data has been properly updated in the database, you need to execute the following commands in MongoDB

1. Use EmployeeDB
2. db.Employee.find({Employeeid :4 })

The first statement ensures that you are connected to the EmployeeDb database. The second statement searches for the record which has the employee id of 4.

1. **Deleting documents in a collection** - Documents can be deleted in a collection using the "deleteOne" method provided by the MongoDB library. The below code snippet shows how to delete a document in a mongoDB collection.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo7.png)

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

db.collection('Employee').deleteOne(

{

"EmployeeName": "Mohan"

}

);

});

**Code Explanation:**

1. Here we are using the "deleteOne" method from the MongoDB library, which is used to delete a document in a mongoDB collection.
2. We are specifying the search criteria of which document needs to be deleted. In our case, we want to find the document which has the EmployeeName of "Mohan" and delete this document.

If you now check the contents of your MongoDB database, you will find the record with Employeeid of 4 and EmployeeName of "Mohan" deleted from the Employee collection.

To check that the data has been properly updated in the database, you need to execute the following commands in MongoDB

1. Use EmployeeDB
2. db.Employee.find()

The first statement ensures that you are connected to the EmployeeDb database. The second statement searches and display all of the records in the employee collection. Here you can see if the record has been deleted or not.

**How to build a node express app with MongoDB to store and serve content**

Building an application with a combination of both using express and MongoDB is quite common nowadays.

When working with[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)web based applications, one will normally here of the term MEAN stack.

* The term MEAN stack refers to a collection of JavaScript based technologies used to develop web applications.
* MEAN is an acronym for MongoDB, ExpressJS,[AngularJS,](https://www.guru99.com/angularjs-tutorial.html)and Node.js.

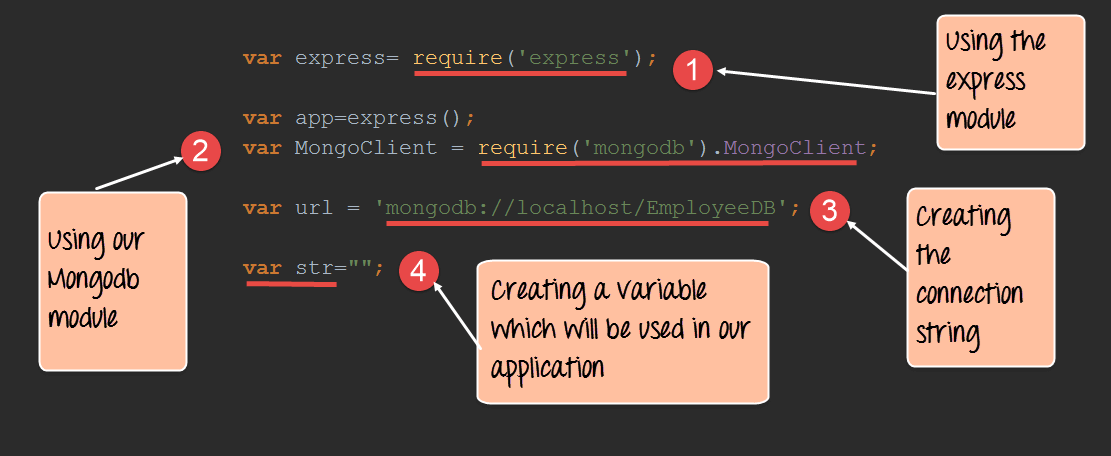
Hence, it's always good to understand how Node.js and MongoDB work together to deliver applications which interact with backend databases.

Let's look at a simple example of how we can use "express" and "MongoDB" together. Our example will make use of the same Employee collection in the MongoDB EmployeeDB database.

We will now incorporate Express to display the data on our web page when it is requested by the user. When our application runs on Node.js, one might need to browse to the URL **http://localhost:3000/Employeeid**.

When the page is launched, all the employee id in the Employee collection will be displayed. So let's see the code snippet in sections which will allow us to achieve this.

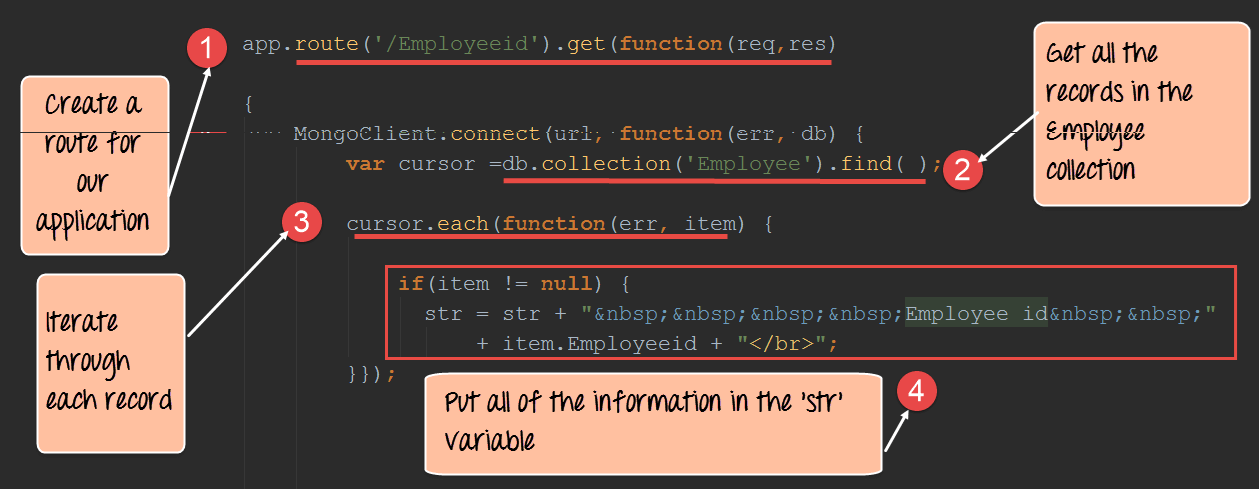
**Step 1)** Define all the libraries which need to be used in our application, which in our case is both the MongoDB and express library.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo8.png)

**Code Explanation:**

1. We are defining our 'express' library, which will be used in our application.
2. We are defining our 'express' library, which will be used in our application for connecting to our MongoDB database.
3. Here we are defining the URL of our database to connect to.
4. Finally, we are defining a string which will be used to store our collection of employee id which need to be displayed in the browser later on.

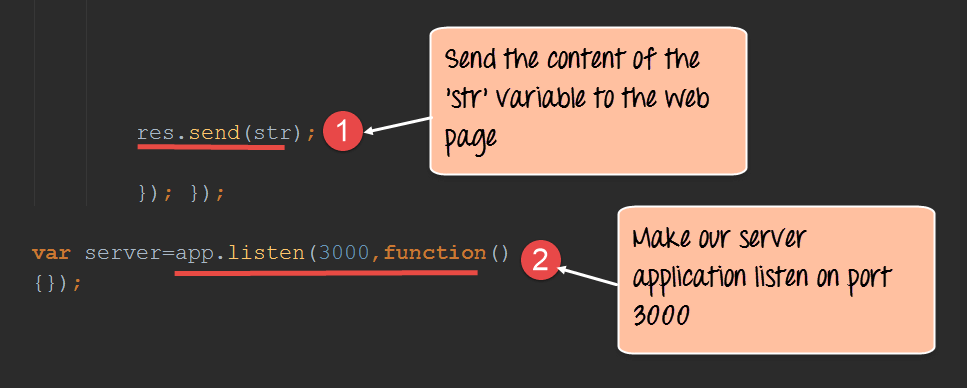
**Step 2)**In this step, we are now going to get all of the records in our 'Employee' collection and work with them accordingly.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo9.png)

**Code Explanation:**

1. We are creating a route to our application called 'Employeeid.' So whenever anybody browses to **http://localhost:3000/Employeeid** of our application, the code snippet defined for this route will be executed.
2. Here we are getting all of the records in our 'Employee' collection through the db.collection('Employee').find() command. We are then assigning this collection to a variable called cursor. Using this cursor variable, we will be able to browse through all of the records of the collection.
3. We are now using the cursor.each() function to navigate through all of the records of our collection. For each record, we are going to define a code snippet on what to do when each record is accessed.
4. Finally, we see that if the record returned is not null, then we are taking the employee via the command "item.Employeeid". The rest of the code is just to construct a proper HTML code which will allow our results to be displayed properly in the browser.

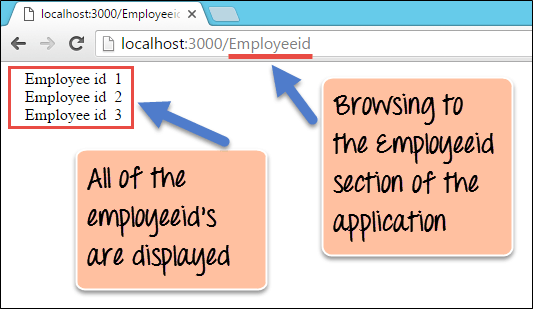
**Step 3)**In this step, we are going to send our output to the web page and make our application listen on a particular port.

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo10.png)

**Code Explanation:**

1. Here we are sending the entire content which was constructed in the earlier step to our web page. The 'res' parameter allows us to send content to our web page as a response.
2. We are making our entire Node.js application listen on port 3000.

**Output:**

[](https://www.guru99.com/images/NodeJS/010716_0632_NodejsMongo11.png)

From the output,

* It clearly shows that all of the employeeid's in the Employee collection were retrieved. This is because we use the MongoDB driver to connect to the database and retrieve all the Employee records and subsequently used "express" to display the records.

Here is the code for your reference

var express = require('express');

var app = express();

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

var str = "";

app.route('/Employeeid').get(function(req, res)

{

MongoClient.connect(url, function(err, db) {

var cursor = db.collection('Employee').find();

//noinspection JSDeprecatedSymbols

cursor.each(function(err, item) {

if (item != null) {

str = str + "    Employee id  " + item.Employeeid + "</br>";

}

});

res.send(str);

db.close();

});

});

var server = app.listen(3000, function() {});

**Note:** cursor.each maybe deprecated based on version of your MongoDB driver. You can append //noinspection JSDeprecatedSymbols before cursor.each to circumvent the issue. Alternatively, you can use forEach. Below is the sample code using forEach

var express = require('express');

var app = express();

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

var str = "";

app.route('/Employeeid').get(function(req, res) {

MongoClient.connect(url, function(err, db) {

var collection = db.collection('Employee');

var cursor = collection.find({});

str = "";

cursor.forEach(function(item) {

if (item != null) {

str = str + " Employee id " + item.Employeeid + "</br>";

}

}, function(err) {

res.send(str);

db.close();

}

);

});

});

var server = app.listen(8080, function() {});

**Summary**

* Node.js is used in conjunction with NoSQL databases to build a lot of modern days web applications. Some of the common databases used are MySQL and MongoDB.
* One of the common modules used for working with MongoDB databases is a module called 'MongoDB.' This module is installed via the Node package manager.
* With the MongoDB module, it's possible to query for records in a collection and perform the normal update, delete and insert operations.
* Finally, one of the modern practices is to use the express framework along with MongoDB to deliver modern-day applications. The Express framework can make use of the data returned by the MongoDB driver and display the data to the user in the web page accordingly.

**Node.js Promise Tutorial**

In previous tutorials, you would have seen callback functions which are used for Asynchronous events. But sometimes callback functions can become a nightmare when they start becoming nested, and the program starts to become long and complex.

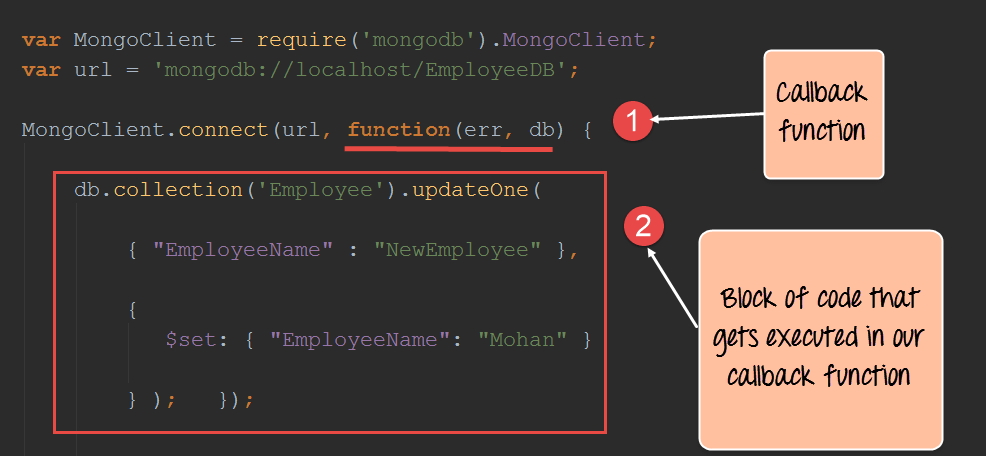
In this tutorial, you will learn-

* [What are promises?](https://www.guru99.com/node-js-promise-generator-event.html#1)
* [Callbacks to promises](https://www.guru99.com/node-js-promise-generator-event.html#2)
* [Dealing with nested promises](https://www.guru99.com/node-js-promise-generator-event.html#3)
* [Creating a custom promise](https://www.guru99.com/node-js-promise-generator-event.html#5)

**What are promises?**

Before we start with promises, let's first revisit what are "callback" functions in Node.js. We have seen these callback functions a lot in the previous chapters, so let's quickly go through one of them.

The example below shows a code snippet, which is used to connect to a[MongoDB](https://www.guru99.com/mongodb-tutorials.html)database and perform an update operation on one of the records in the database.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor1.png)

1. In the above code, the part of the function(err,db) is known as the declaration of an anonymous or callback function. When the MongoClient creates a connection to the MongoDB database, it will return to the callback function once the connection operation is completed. So in a sense, the connection operations happen in the background, and when it is done, it calls our callback function. Remember that this is one of the key points of Node.js to allow many operations to happen concurrently and thus not block any user from performing an operation.
2. The second code block is what gets executed when the callback function is actually called. The callback function just updates one record in our MongoDB database.

So what is a promise then? Well, a promise is just an enhancement to callback functions in Node.js. During the development lifecycle, there may be an instance where you would need to nest multiple callback functions together. This can get kind of messy and difficult to maintain at a certain point in time. In short, a promise is an enhancement to callbacks that looks towards alleviating these problems.

The basic syntax of a promise is shown below;

var promise = doSomethingAync()

promise.then(onFulfilled, onRejected)

* "doSomethingAync" is any callback or asynchronous function which does some sort of processing.
* This time, when defining the callback, there is a value which is returned called a "promise."
* When a promise is returned, it can have 2 outputs. This is defined by the 'then clause'. Either the operation can be a success which is denoted by the 'onFulfilled' parameter. Or it can have an error which is denoted by the 'onRejected' parameter.

**Note:** So the key aspect of a promise is the return value. There is no concept of a return value when working with normal callbacks in Node.js. Because of the return value, we have more control of how the callback function can be defined.

In the next topic, we will see an example of promises and how they benefit from callbacks.

**Callbacks to promises**

Now let's look at an example of how we can use "promises" from within a Node.js application. In order to use promises in a Node.js application, the 'promise' module must first be downloaded and installed.

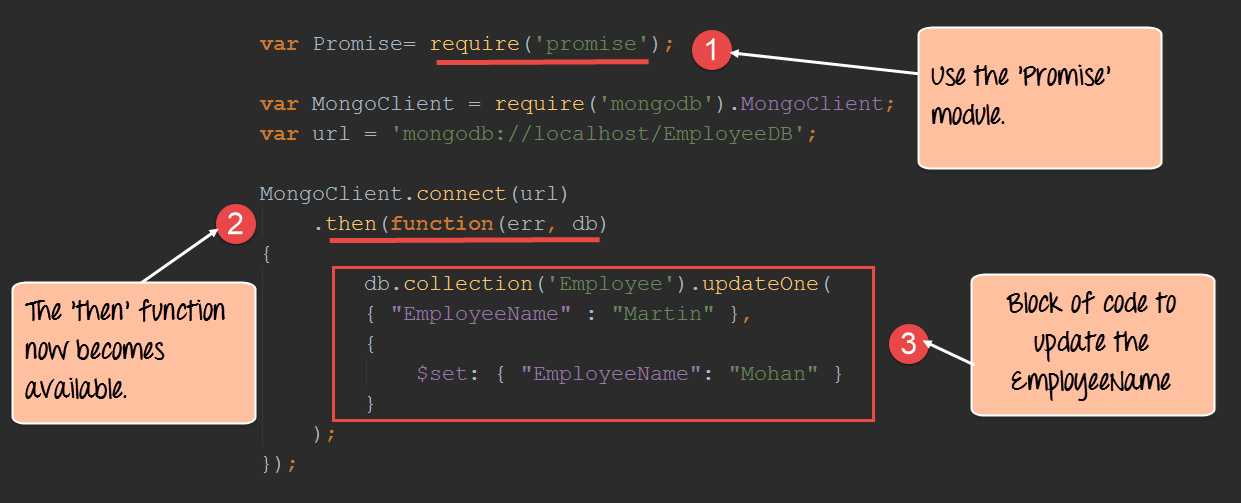
We will then modify our code as shown below, which updates an Employeename in the 'Employee' collection by using promises.

**Step 1)** Installing the NPM Modules

To use Promises from within a Node JS application, the promise module is required. To install the promise module, run the below command

**npm install promise**

**Step 2)**Modify the code to include promises

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor2.png)

var Promise = require('promise');

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url)

.then(function(err, db) {

db.collection('Employee').updateOne({

"EmployeeName": "Martin"

}, {

$set: {

"EmployeeName": "Mohan"

}

});

});

**Code Explanation:-**

1. The first part is to include the 'promise' module which will allow us to use the promise functionality in our code.
2. We can now append the 'then' function to our MongoClient.connect function. So what this does is that when the connection is established to the database, we need to execute the code snippet defined thereafter.
3. Finally, we define our code snippet which does the work of updating EmployeeName of the employee with the name of "Martin" to "Mohan".

**Note:-**

If you now check the contents of your MongoDB database, you will find that if a record with EmployeeName of "Martin" exists, it will be updated to "Mohan."

To check that the data has been properly inserted in the database, you need to execute the following commands in MongoDB

1. Use EmployeeDB
2. db.Employee.find({EmployeeName :Mohan })

The first statement ensures that you are connected to the EmployeeDb database. The second statement searches for the record which has the employee name of "Mohan".

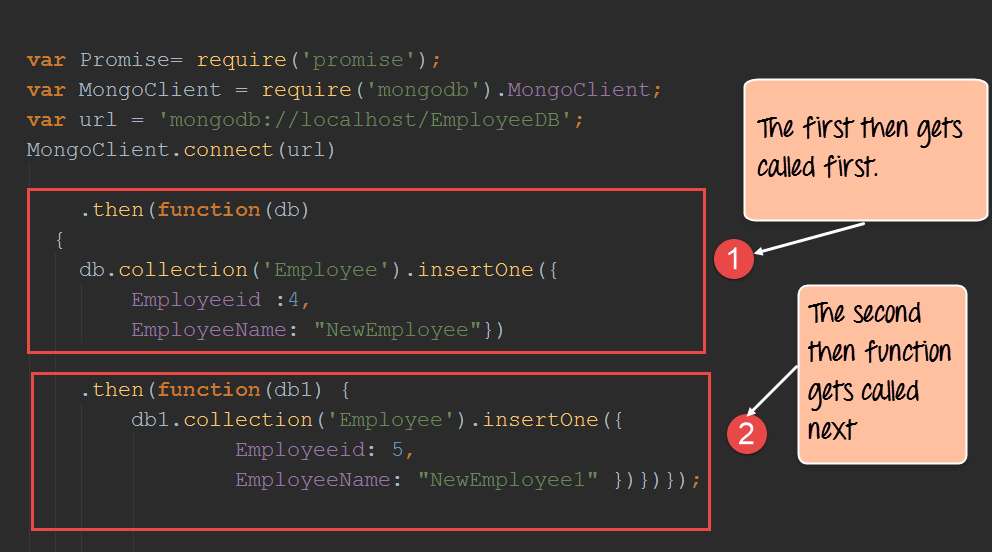
**Dealing with nested promises**

When defining promises, it needs to be noted that the "then" method itself returns a promise. So in a sense, promises can be nested or chained to each other.

In the example below, we use chaining to define 2 callback functions, both of which insert a record into the MongoDB database.

(**Note**: Chaining is a concept used to link execution of methods to one another. Suppose if your application had 2 methods called 'methodA' and 'methodB.' And the logic was such that 'methodB' should be called after 'methodA,' then you would chain the execution in such a way that 'methodB' gets called directly after 'methodA.')

The key thing to note in this example is that the code becomes cleaner, readable and maintainable by using nested promises.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor3.png)

var Promise = require('promise');

var MongoClient = require('mongodb').MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url)

.then(function(db) {

db.collection('Employee').insertOne({

Employeeid: 4,

EmployeeName: "NewEmployee"

})

.then(function(db1) {

db1.collection('Employee').insertOne({

Employeeid: 5,

EmployeeName: "NewEmployee1"

})

})

});

**Code Explanation:-**

1. We are now defining 2 "then" clauses which get executed one after the other. In the first then clause, we are passing the 'db' parameter which contains our database connection. We are then using the collection property of the 'db' connection to insert records into the 'Employee' collection. The 'insertOne' method is used to insert the actual document into the Employee collection.
2. We are then using the 2nd then clause also to insert another record into the database.

If you now check the contents of your MongoDB database, you will find the 2 record's inserted into the MongoDB database.

**Creating a custom promise**

A custom promise can be created by using a node module called 'q.' The 'q' library needs to be downloaded and installed using the node package manager. After using the 'q' library, the method "denodeify" can be called which will cause any function to become a function which returns a promise.

In the example below, we will create a simple function called "Add" which will add 2 numbers. We will convert this function into a function to return a promise.

Once that is done, we will use the promise returned by the Add function to display a message in the console.log.

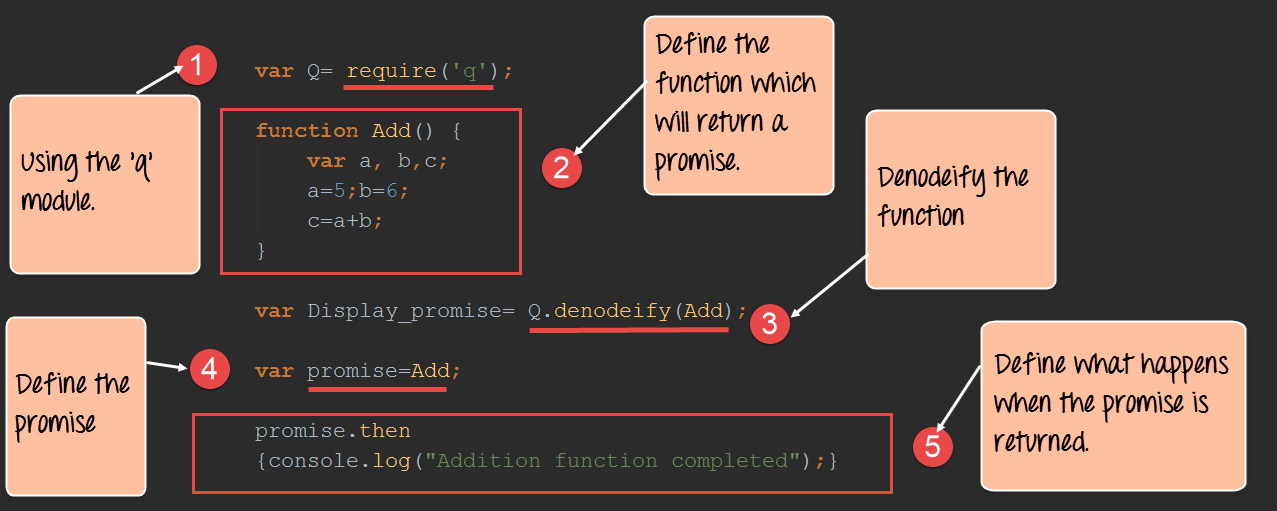
Let's follow the below steps to creating our custom function to return a promise.

**Step 1)** Installing the NPM Modules

To use 'q' from within a Node JS application, the 'q' module is required. To install the 'q' module, run the below command

**npm install q**

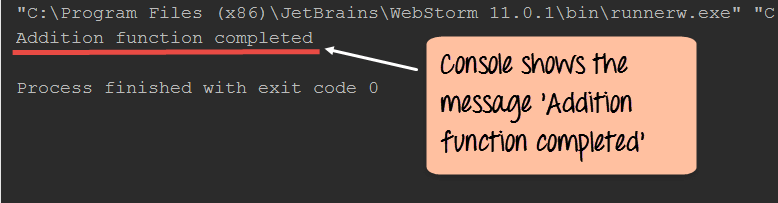
**Step 2)** Define the following code which will be used to create the custom promise.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor7.png)

**Code Explanation:-**

1. The first bit is to include the 'q' library by using the require keyword. By using this library, we will be able to define any function to return a callback.
2. We are creating a function called Add which will add 2 numbers defined in variables a and b. The sum of these values will be stored in variable c.
3. We are then using the q library to denodeify ( the method used to convert any function into a function that would return a promise) our Add function or in otherwise convert our Add function to a function which returns a promise.
4. We now call our "Add" function and are able to get a return promise value because of the prior step we performed of denodeify the Add function.
5. The 'then' keyword is used specify that if the function is executed successfully then display the string "Addition function completed" in the console.log.

When the above code is run, the output "Addition function completed" will be displayed in the console.log as shown below.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor8.png)

**Summary**

* Using callback functions in Node.js does have its disadvantages. Sometimes during the process of development, the nested use of callback functions can make the code messier and difficult to maintain.
* Most of the issues with nested callback functions can be mitigated with the use of promises and generators in node.js
* A Promise is a value returned by an asynchronous function to indicate the completion of the processing carried out by the asynchronous function.
* Promises can be nested within each other to make code look better and easier to maintain when an asynchronous function need to be called after another asynchronous function

**Bluebird Promises Tutorial**

Bluebird is a fully-featured Promise library for JavaScript. The strongest feature of Bluebird is that it allows you to "promisify" other Node modules in order to use them asynchronously. Promisify is a concept applied to callback functions. This concept is used to ensure that every callback function which is called returns some value.

So if a Node JS module contains a callback function which does not return a value, and if we Promisify the node module, all the function's in that specific node module would automatically be modified to ensure that it returns a value.

So you can use BlueBird to make the[MongoDB](https://www.guru99.com/mongodb-tutorials.html)module run asynchronously. This just adds another level of ease when writing Node.js applications.

We will look at an example of how to use the bluebird module.

Our example will first establish a connection to the "Employee collection" in the "EmployeeDB" database. If "then" connection is established, then it will get all of the records in the collection and display them in the console accordingly.

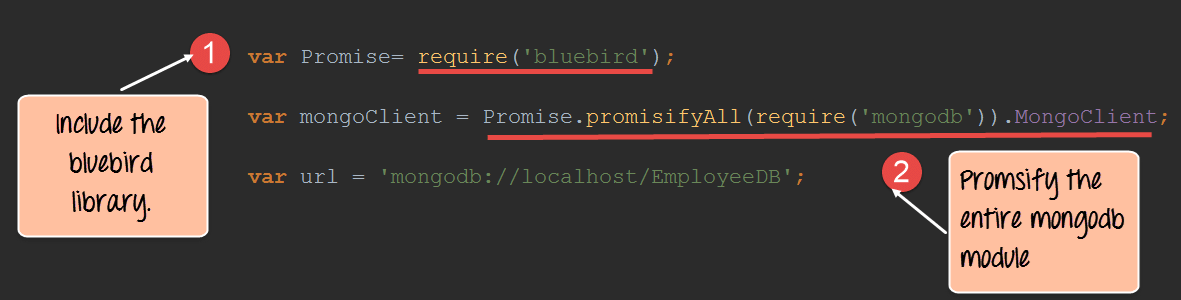
**Generating promises with the BlueBird library**

**Step 1)** Installing the NPM Modules

To use Bluebird from within a Node application, the Bluebird module is required. To install the Bluebird module, run the below command

**npm install bluebird**

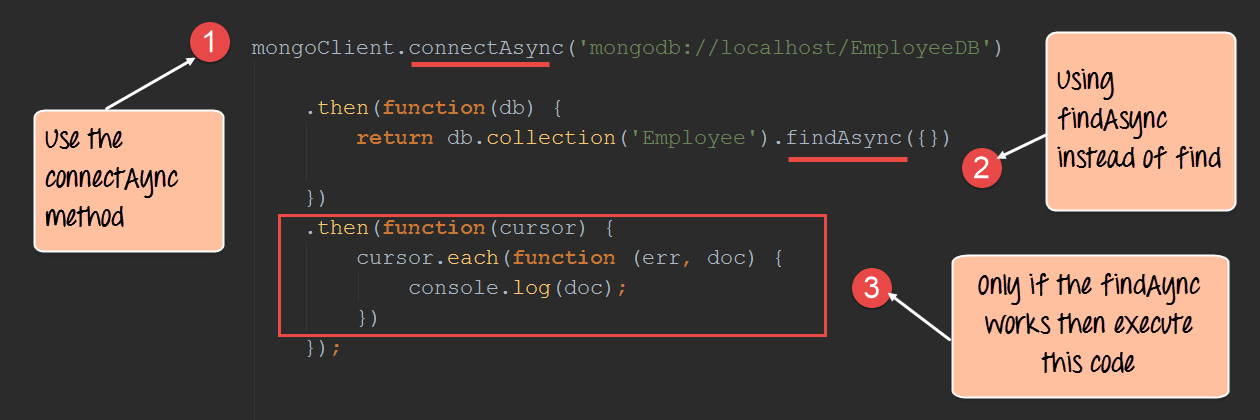
**Step 2)**The next step is to include the bluebird module in your code and promisify the entire MongoDB module. By promisify, we mean that bluebird will ensure that each and every method defined in the MongoDB library returns a promise.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor4.png)

**Code Explanation:-**

1. The require command is used to include the Bluebird library.
2. Use Bluebird's .promisifyAll() method to create an async version of every method the MongoDB module provides. This ensures that each method of the MongoDB module will run in the background and ensure that a promise is returned for each method call in the MongoDB library.

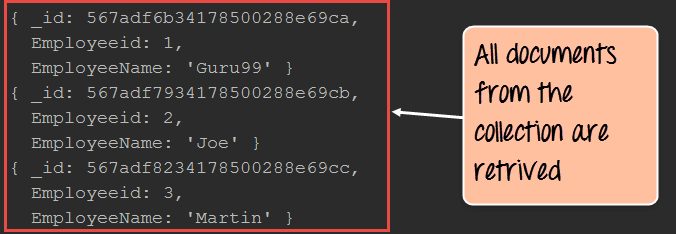
**Step 3)** The final step is to connect to our database, retrieve all the records in our collection and display them in our console log.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor5.png)

**Code Explanation:-**

1. You will notice that we are using the "connectAsync" method instead of the normal connection method for connecting to the database. Bluebird actually adds the Async keyword to each method in the MongoDB library to distinguish those calls which return promises and those which don't. So there is no guarantee that methods without the Async word will return a value.
2. Similar to the connectAsync method, we are now using the findAsync method to return all of the records in the mongoDB 'Employee' collection.
3. Finally, if the findAsync returns a successful promise we then define a block of code to iterate through each record in the collection and display them in the console log.

If the above steps are carried out properly, all of the documents in the Employee collection will be displayed in the console as shown in the output below.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor6.png)

Here is the code for your reference

var Promise = require('bluebird');

var mongoClient = Promise.promisifyAll(require('mongodb')).MongoClient;

var url = 'mongodb://localhost/EmployeeDB';

mongoClient.connectAsync('mongodb://localhost/EmployeeDB')

.then(function(db) {

return db.collection('Employee').findAsync({})

})

.then(function(cursor) {

cursor.each(function(err, doc) {

console.log(doc);

})

});

**Node.js Generators & Compare with Callbacks**

In this tutorial, we are going to learn about Generators and their differences with Callbacks

**What are generators?**

Generators have become quite famous in Node.js in recent times and that probably because of what they are capable of doing.

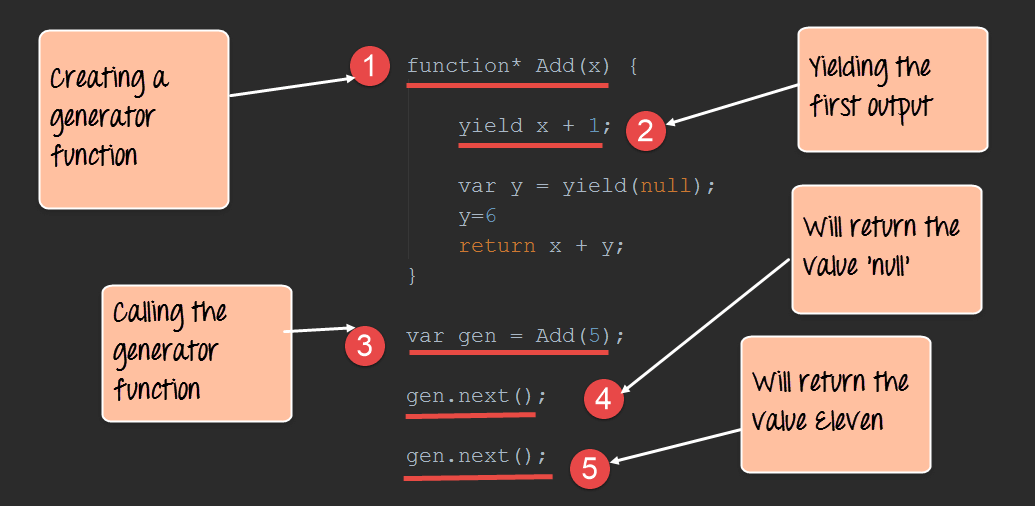
* Generators are function executions that can be suspended and resumed at a later point.
* Generators are useful when carrying out concepts such as 'lazy execution'. This basically means that by suspending execution and resuming at will, we are able to pull values only when we need to.

Generators have the below 2 key methods

1. Yield method – The yield method is called in a function to halt the execution of the function at the specific line where the yield method is called.
2. Next method – This method is called from the main application to resume the execution of a function which has a yield method. The execution of the function will continue till the next yield method or till the end of the method.

Let's look at an example of how generators can be used.

In our example, we are going to have a simple Add function which will add 2 numbers, but we will keep on halting the method execution at different points to showcase how generators can be used.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor9.png)

function\* Add(x) {

yield x + 1;

var y = yield(null);

y = 6

return x + y;

}

var gen = Add(5);

gen.next();

gen.next();

**Code Explanation:-**

1. The first step is to define our generator "function". Note that this is done by adding a "\*" to the function keyword. We are then defining a function called Add which takes a parameter of x.
2. The yield keyword is a specific to generators. This makes it a powerful construct for pausing a function in the middle of anything. So here, the function execution will be halted till we invoke the next() function, which will be done in Step4. At this point, the value of x will become 6 and the execution of the function will be stopped.
3. This is where we first call the generator function and send the value of 5 to our Add function. This value will be substituted in the x parameter of our Add function.
4. Once we call the next() function, the Add() function will resume the execution. When the next statement var y= yield(null) will be executed, the Add() function will again stop executing.
5. Now after calling the next() function again, the next statements will run, and the combined value of x=5 and y=6 will be added and returned.

**Callbacks vs. generators**

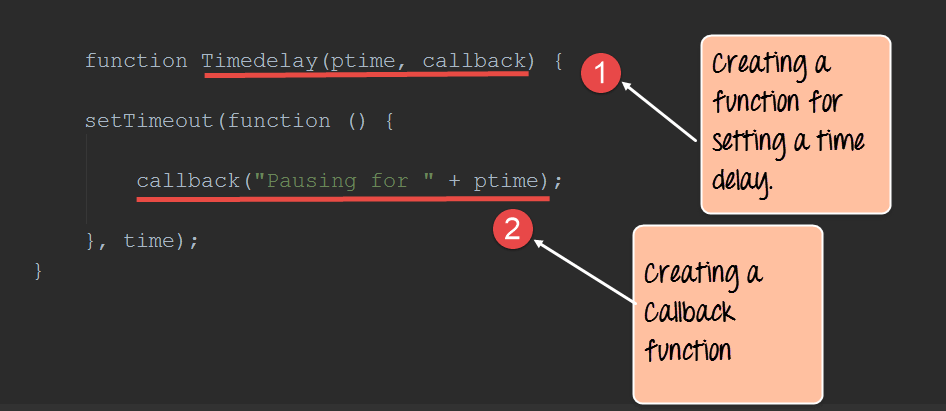
Generators are used to solve the problem of what is known as callback hell. Sometimes callback functions become so nested during the development of a Node.js application that it just becomes too complicated to use callback functions.

This is where generators are useful. One of the most common examples of this is when creating timer functions.

Let's see the below example of how generators can prove to be useful over callbacks.

Our example will just create a simple time delay function. We would then want to call this function incorporating a delay of 1000, 2000 and 3000 ms.

**Step 1)** Define our callback function with the necessary time delay code.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor10.png)

function Timedelay(ptime, callback) {

setTimeout(function() {

callback("Pausing for " + ptime);

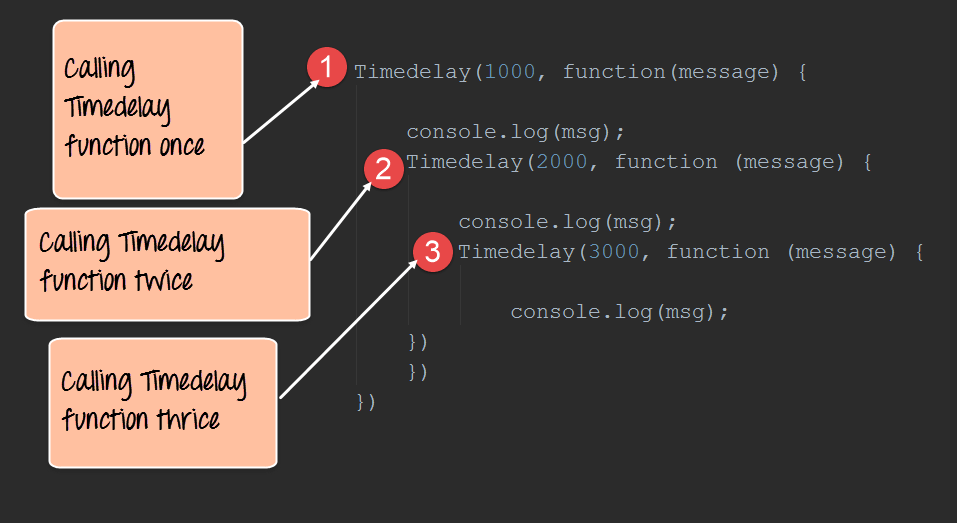
}, time);

}

**Code Explanation:-**

1. Here we are creating a function called Timedelay with a parameter called ptime. This will take in the necessary time delay we want to introduce in our application.
2. The next step is to just create a message, which will be displayed to the user saying that the application is going to be pause for these many numbers of milliseconds.

**Step 2)** Now let's look at the code if we were incorporating callbacks. Suppose we wanted to incorporate callbacks based on the value of 1000, 2000 and 3000 milliseconds, the below code shows how we would need to implement these using callbacks.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor11.png)

Timedelay(1000, function(message) {

console.log(msg);

Timedelay(2000, function(message) {

console.log(msg);

Timedelay(3000, function(message) {

console.log(msg);

})

})

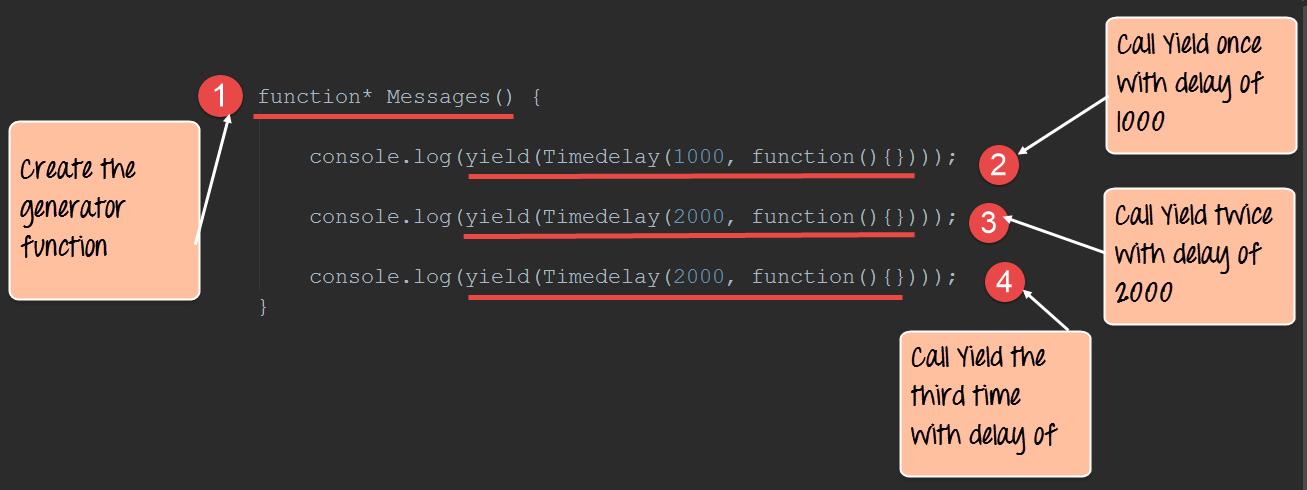
})

**Code Explanation:-**

1. We are calling the Timedelay as a callback with 1000 as the value.
2. Next we want to call the Timedelay function again with 2000 as the value.
3. Finally, we want to call the Timedelay function again with 3000 as the value.

From the above code, you can see that it becomes messier as we want to start calling the function multiple times.

**Step 3)** Now let's see how to implement the same code using generators. From the below code you can now see how simple it has become to implement the Timedelay function using generators.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor12.png)

function\* Messages() {

console,log(yield(Timedelay(1000, function(){})));

console,log(yield(Timedelay(2000, function(){})));

console,log(yield(Timedelay(3000, function(){})));

}

**Code Explanation:-**

1. We are first defining a generator function which will be used to call our Timedelay function.
2. We are calling the Yield function along with the Timedelay function with 1000 as the parameter value.
3. We are then calling the Yield function along with the Timedelay function with 2000 as the parameter value.
4. Finally, we are calling the Yield function along with the Timedelay function with 3000 as the parameter value.

**Summary**

Generators can also be used to alleviate the problems with nested callbacks and assist in removing what is known as the callback hell. Generators are used to halt the processing of a function. This is accomplished by usage of the 'yield' method in the asynchronous function.

**Node js Streams Tutorial: Filestream, Pipes**

In this tutorial, you will learn

* [Filestream in Node.js](https://www.guru99.com/node-js-streams-filestream-pipes.html#8)
* [Pipes in Node.js](https://www.guru99.com/node-js-streams-filestream-pipes.html#9)
* [Events in Node.js](https://www.guru99.com/node-js-streams-filestream-pipes.html#10)
* [Emitting Events](https://www.guru99.com/node-js-streams-filestream-pipes.html#11)

**Filestream in Node.js**

Node makes extensive use of streams as a data transfer mechanism.

For example, when you output anything to the console using the console.log function, you are actually using a stream to send the data to the console.

Node.js also has the ability to stream data from files so that they can be read and written appropriately. We will now look at an example of how we can use streams to read and write from files. We need to follow the below-mentioned steps for this example

**Step 1)** Create a file called data.txt which has the below data. Let assume this file is stored on the D drive of our local machine.

Tutorial on Node.js

Introduction

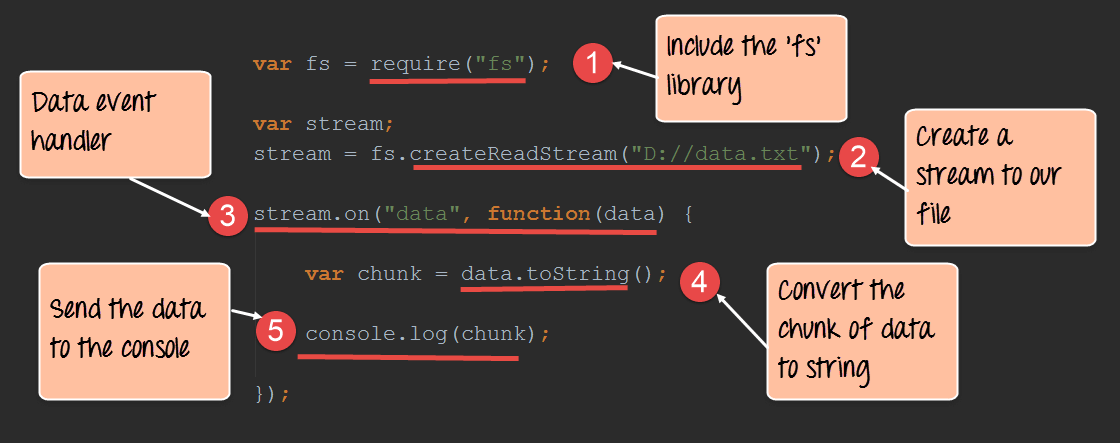
Events

Generators

Data Connectivity

Using Jasmine

**Step 2)** Write the relevant code which will make use of streams to read data from the file.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor13.png)

var fs = require("fs");

var stream;

stream = fs.createReadStream("D://data.txt");

stream.on("data", function(data) {

var chunk = data.toString();

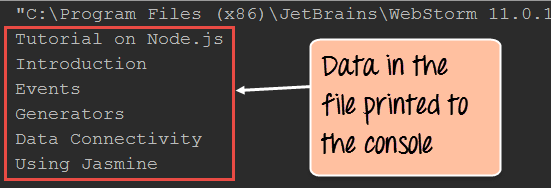
console.log(chunk);

});

**Code Explanation:-**

1. We first need to include the 'fs' modules which contain all the functionality required to create streams.
2. Next we create a readable stream by using the method – createReadStream. As an input, we give the location of our data.txt file.
3. The steam.on function is an event handler and in it, we are specifying the first parameter as 'data.' This means that whenever data comes in the stream from the file, then execute a callback function. In our case, we are defining a callback function which will carry out 2 basic steps. The first is to convert the data read from the file as a string. The second would be to send the converted string as an output to the console.
4. We are taking each chunk of data which is read from the data stream and converting it to a string.
5. Finally, we are sending the output of each string converted chunk to the console.

**Output:**

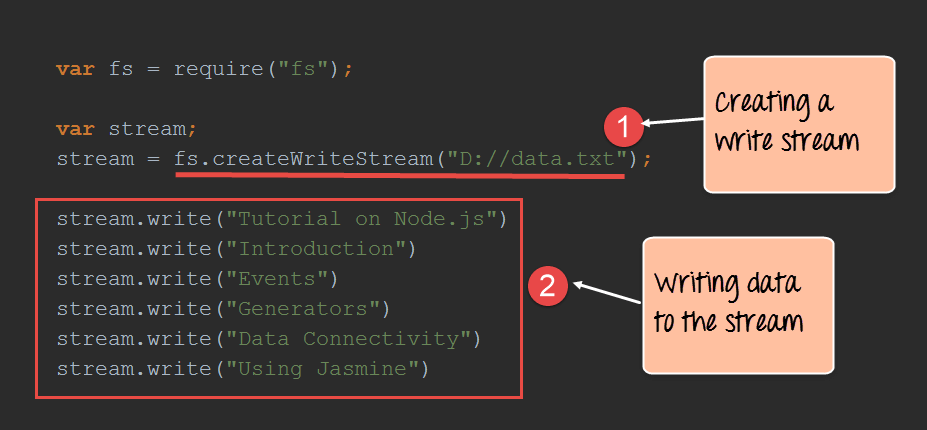
[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor14.png)

* If the code is executed properly, you will see the above output in the console. This output will be the same as that in the data.txt file.

**Writing to a file**

In the same way, that we create a read stream, we can also create a write stream to write data to a file. Let's first create an empty file with no contents called data.txt. Let's assume this file is placed in the D drive of our computer.

The below code shows how we can write data to the file.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor15.png)

var fs = require("fs");

var stream;

stream = fs.createWriteStream("D://data.txt");

stream.write("Tutorial on Node.js")

stream.write("Introduction")

stream.write("Events")

stream.write("Generators")

stream.write("Data Connectivity")

stream.write("Using Jasmine")

**Code Explanation:-**

1. We are creating a writable stream by using the method – createWriteStream. As an input, we give the location of our data.txt file.
2. Next we used the stream.write a method to write the different lines of text to our text file. The stream will take care of writing this data to the data.txt file.

If you open the data.txt file, you will now see the following data in the file

Tutorial on Node.js

Introduction

Events

Generators

Data Connectivity

Using Jasmine

**Pipes in Node.js**

Within Node applications, streams can be piped together using the pipe() method, which takes two arguments:

* A Required writable stream that acts as the destination for the data and
* An optional object used to pass in options.

A typical example of using pipes, if you want to transfer data from one file to the other.

So let's see an example of how we can transfer data from one file to the other using pipes.

**Step 1)** Create a file called datainput.txt which has the below data. Let assume this file is stored on the D drive of our local machine.

Tutorial on Node.js

Introduction

Events

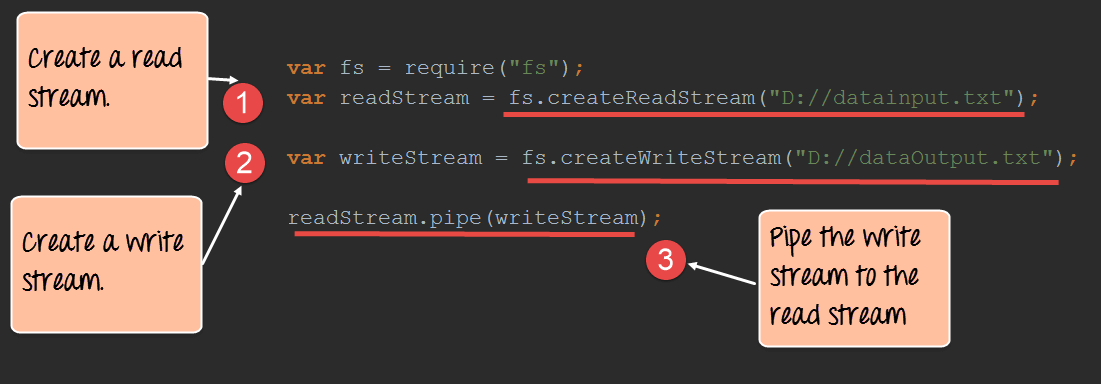
Generators

Data Connectivity

Using Jasmine

**Step 2)** Create a blank empty file called dataOutput.txt and placed it on the D drive of your local machine.

**Step 3)** Write the below code to carry out the transfer of data from the datainput.txt file to the dataOutput.txt file.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor16.png)

var fs = require("fs");

var readStream = fs.createReadStream("D://datainput.txt");

var writeStream = fs.createWriteStream("D://dataOutput.txt");

readStream.pipe(writeStream);

**Code Explanation:-**

1. We are first creating a "readstream" to our datainput.txt file which contains all our data which needs to be transferred to the new file.
2. We then need to create a "writestream" to our dataOutput.txt file, which is our empty file and is the destination for the transfer of data from the datainput.txt file.
3. We then use the pipe command to transfer the data from the readstream to the write stream. The pipe command will take all the data which comes into the readstream, and push it to the writestream.

If you now open the dataOutput.txt file, you will see all the data which was present in the datainput.txt file.

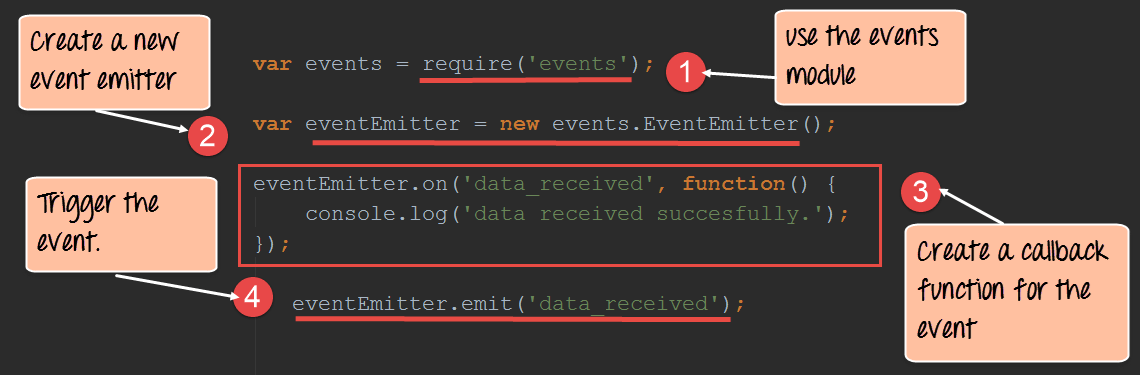
**Events in Node.js**

Events are one of the key concepts in Node.js and sometimes Node.js is referred to as an Event-driven framework.

Basically, an event is something that happens. For example, if a connection is established to a database, then the database connection event is triggered. Event driven programming is to create functions that will be triggered when specific events are triggered.

Let's look at a basic example of defining an event in Node.js.

We are going to create an event called 'data\_received'. When this event is triggered, the text "data received" will be sent to the console.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor17.png)

var events = require('events');

var eventEmitter = new events.EventEmitter();

eventEmitter.on('data\_received', function() {

console.log('data received succesfully.');

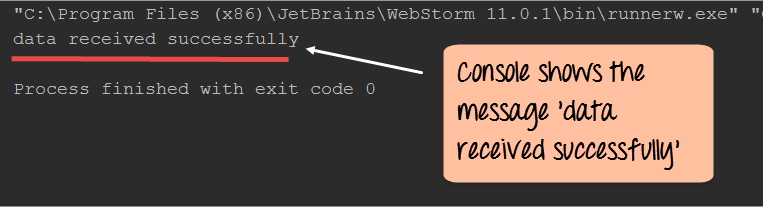
});

eventEmitter.emit('data\_received');

**Code Explanation:-**

1. Use the require function to include the 'events' module. With this module, you will be able to create events in Node.js.
2. Create a new events emitter. This is used to bind the event, which in our case is "data\_received" to a callback function which is defined in step3.
3. We define an event-driven function which says that if in case the "data\_received" event is triggered then we should output the text "data\_received" to the console.
4. Finally, we do have a manual trigger of our event using the eventEmiter.emit function. This will trigger the data\_received event.

When the program is run, the text "data received" will be sent to the console as shown below.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor18.png)

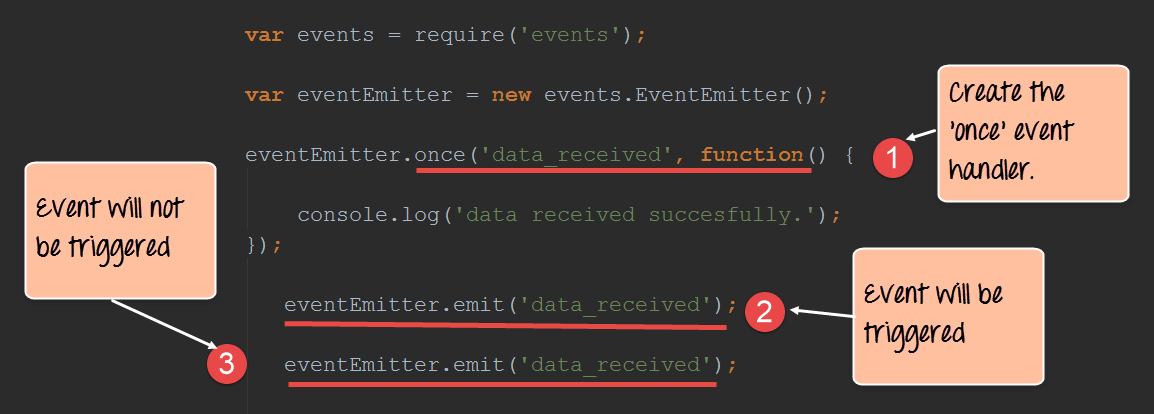
**Emitting Events**

When defining events, there are different methods for events which can be invoked. This topic focuses on looking at each one of them in detail.

1. **One time event handlers**

Sometimes you may be interested in reacting to an event only the first time it occurs. In these situations, you can use the once() method.

Let's see how we can make use of the once method for event handlers.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor19.png)

**Code Explanation:-**

1. Here we are using the 'once' method to say that for the event 'data\_received,' the callback function should only be executed once.
2. Here we are manually triggering the 'data\_received' event.
3. When the 'data\_received' event is triggered again, this time, nothing will happen. This is because of the first step where we said that the event could only be triggered once.

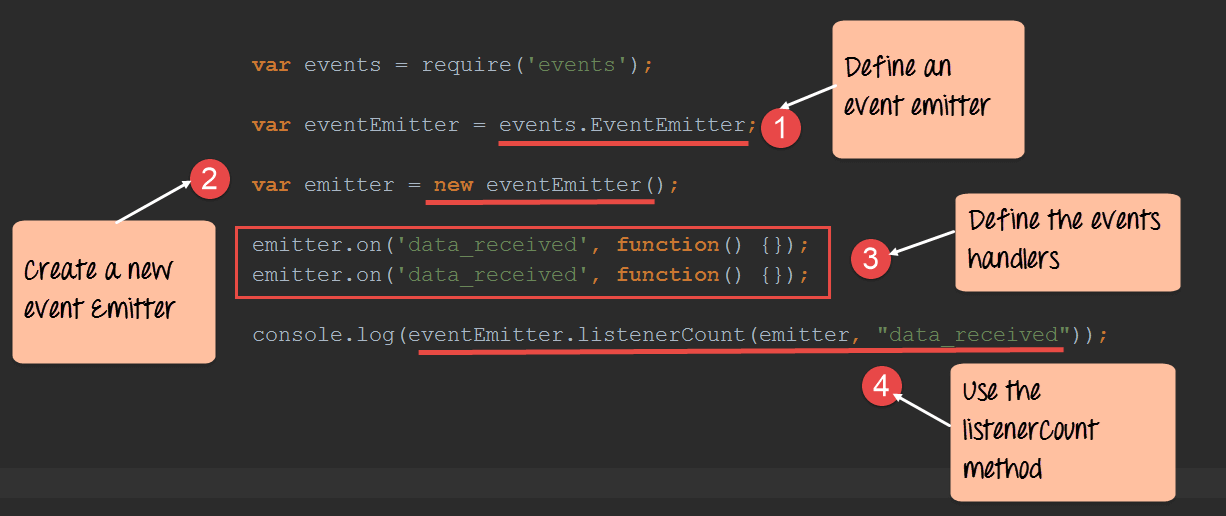
If the code is executed properly, the output in the log will be 'data\_received successfully'. This message will only appear once in the console.

1. **Inspecting Event Listeners**

At any point in its lifetime, an event emitter can have zero or more listeners attached to it. The listeners for each event type can be inspected in several ways.

If you are interested in only determining the number of attached listeners, then look no further than the EventEmitter.listenerCount() method.

(**Note:** Listeners are important because the main program should know if listeners are being added on the fly to an event, else the program will malfunction because additional listeners will get called.)

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor20.png)

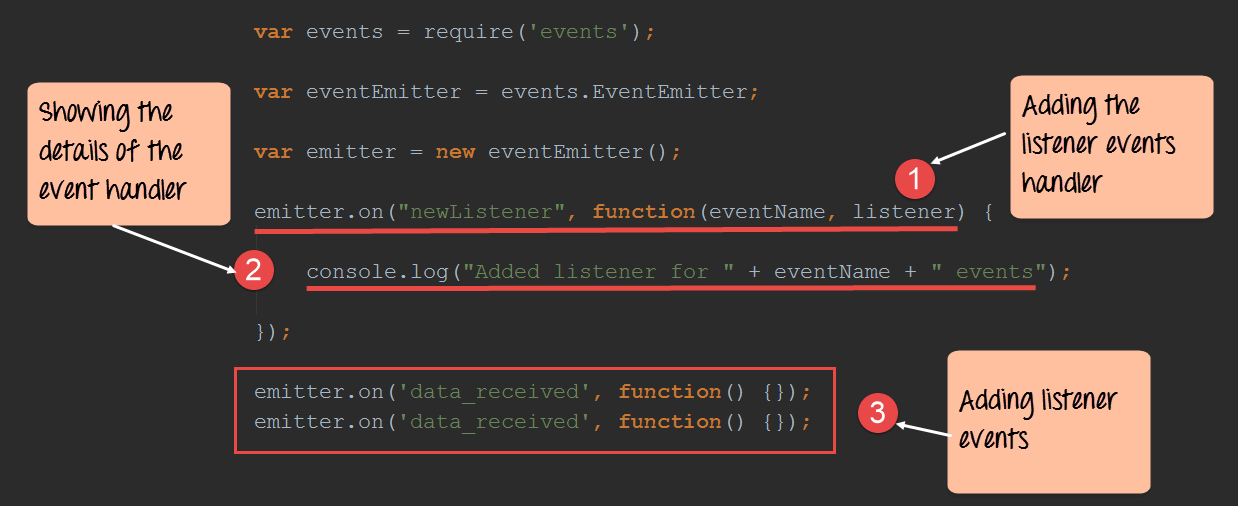
**Code Explanation:-**

1. We are defining an eventEmitter type which is required for using the event-related methods.
2. We are then defining an object called emitter which will be used to define our event handlers.
3. We are creating 2 events handlers which basically do nothing. This is kept simple for our example just to show how the listenerCount method works.
4. Now when you invoke the listenerCount method on our data\_received event, it will send the number of event listeners attached to this event in the console log.

If the code is executed properly, the value of 2 will be shown in the console log.

1. **The newListener Event**

Each time a new event handler is registered, the event emitter emits a newListener event. This event is used to detect new event handlers. You typically use newListener event when you need to allocate resources or perform some action for each new event handler.

[](https://www.guru99.com/images/NodeJS/010716_0659_NodejsTutor21.png)

var events = require('events');

var eventEmitter = events.EventEmitter;

var emitter = new eventEmitter();

emitter.on("newListener", function(eventName, listener) {

console.log("Added listener for " + eventName + " events");

});

emitter.on('data\_received', function() {});

emitter.on('data\_received', function() {});

**Code Explanation:-**

1. We are creating a new event handler for the 'newListener' event. So whenever a new event handler is registered, the text "Added listener for" + the event name will be displayed in the console.
2. Here we are writing to the console the text "Added listener for" + the event name for each event registered.
3. We are defining 2 event handlers for our event 'data\_received'.

If the above code is executed properly, the below text will be shown in the console. It just shows that the 'newListener' event handler was triggered twice.

Added listener for data\_received events

Added listener for data\_received events

**Summary**

* Streams are used in Node.js to read and write data from Input-Output devices. Node.js makes use of the 'fs' library to create readable and writable streams to files. These streams can be used to read and write data from files.
* Pipes can be used to connect multiple streams together. One of the most common example is to pipe the read and write stream together for the transfer of data from one file to the other.
* Node.js is often also tagged as an event driven framework, and it's very easy to define events in Node.js. Functions can be defined which respond to these events.
* Events also expose methods for responding to key events. For example, we have seen the once() event handler which can be used to make sure that a callback function is only executed once when an event is triggered.

**Node.js Unit Testing Tutorial with Jasmine**

Testing is a key element to any application. For Node.js, the framework available for Testing is called Jasmine. In early 2000, there was a framework for testing[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)applications called JsUnit. Later this framework got upgraded and is now known as Jasmine.

Jasmine helps in automated [Unit Testing](https://www.guru99.com/unit-testing-guide.html), something which has become quite a key practice when developing and deploying modern-day web applications.

In this tutorial, you will learn how to get your environment setup with jasmine and how you can start testing your first Node.js application with jasmine.

In this tutorial, you will learn-

* [Overview of Jasmine for testing Node.js applications](https://www.guru99.com/node-js-testing-jasmine.html#1)
* [How to use Jasmine to test Node.js applications](https://www.guru99.com/node-js-testing-jasmine.html#2)

**Overview of Jasmine for testing Node.js applications**

Jasmine is a **Behavior Driven Development(BDD)** testing framework for JavaScript. It does **not** rely on browsers, DOM, or any JavaScript framework. Thus, it's suited for websites, Node.js projects, or anywhere that JavaScript can run. To start using Jasmine, you need to first download and install the necessary Jasmine modules.

Next, you would need to initialize your environment and inspect the jasmine configuration file. The below steps shows how to setup Jasmine in your environment

**Step 1)** Installing the NPM Modules

You to need to install jasmine module to use the jasmine framework from within a Node application. To install the jasmine-node module, run the below command.

**npm install jasmine-node**

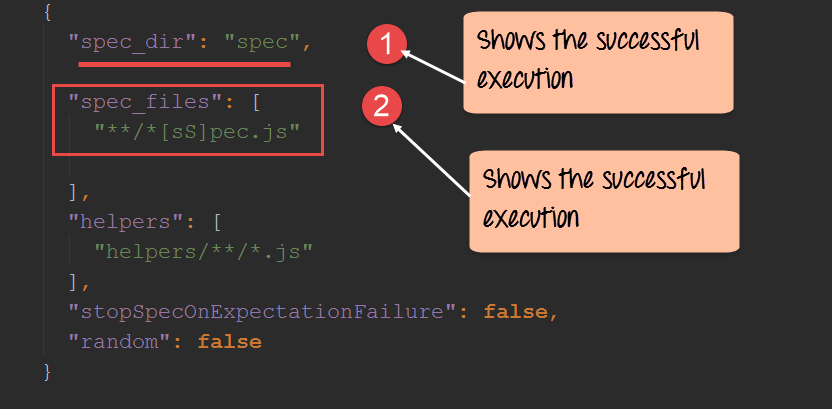
**Step 2)** Initializing the project – By doing this, jasmine creates a spec directory and configuration json for you. The spec directory is used to store all your test files. By doing this, jasmine will know where all your tests are, and then can execute them accordingly. The JSON file is used to store specific configuration information about jasmine.

To initialize the jasmine environment, run the below command

**jasmine init**

**Step 3)** Inspect your configuration file. The configuration file will be stored in the spec/support folder as jasmine.json. This file enumerates the source files and spec files you would like the Jasmine runner to include.

The below screenshot shows a typical example of the package.json file for jasmine.

[](https://www.guru99.com/images/NodeJS/010716_0733_NodejsTesti1.png)

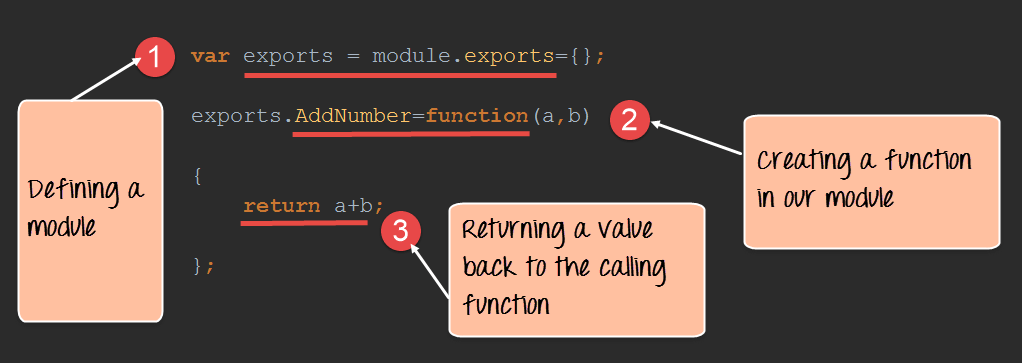
1. Note that the spec directory is specified here. As noted earlier, when jasmine runs, it searches for all tests in this directory.
2. The next thing to note is the spec\_files parameter – This denotes that whatever test files are created they should be appended with the 'spec' keyword.

**How to use Jasmine to test Node.js applications**

In order to use Jasmine to test Node.js applications, a series of steps need to be followed.

In our example below, we are going to define a module which adds 2 numbers which need to be tested. We will then define a separate code file with the test code and then use jasmine to test the Add function accordingly.

**Step 1)** Define the code which needs to be tested. We are going to define a function which will add 2 numbers and return the result. This code is going to be written in a file called "Add.js."

[](https://www.guru99.com/images/NodeJS/010716_0733_NodejsTesti2.png)

var exports=module.exports={};

exports.AddNumber=function(a,b)

{

return a+b;

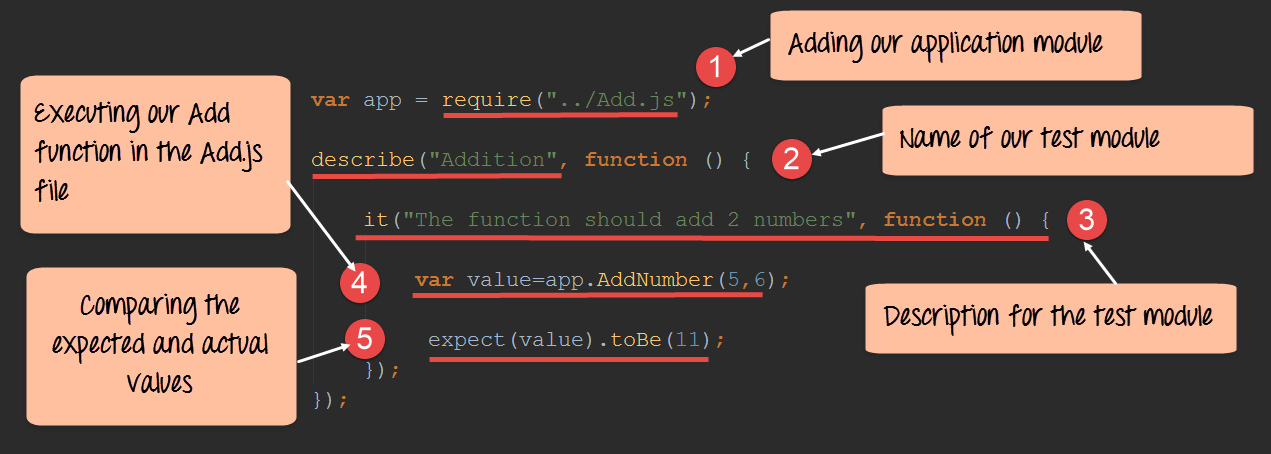
};

**Code Explanation:**

1. The "exports" keyword is used to ensure that the functionality defined in this file can actually be accessed by other files.
2. We are then defining a function called 'AddNumber.' This function is defined to take 2 parameters, a and b. The function is added to the module "exports" to make the function as a public function that can be accessed by other application modules.
3. We are finally making our function return the added value of the parameters.

**Step 2)** Next, we need to define our jasmine test code, which will be used to test our "Add" function In the Add.js file. The below code needs to put in a file called **add-spec.js.**

**Note**: - The word 'spec' needs to be added to the test file so that it can be detected by jasmine.

[](https://www.guru99.com/images/NodeJS/010716_0733_NodejsTesti3.png)

**Code Explanation:**

var app=require("../Add.js");

describe("Addition",function(){

it("The function should add 2 numbers",function() {

var value=app.AddNumber(5,6);

expect(value).toBe(11);

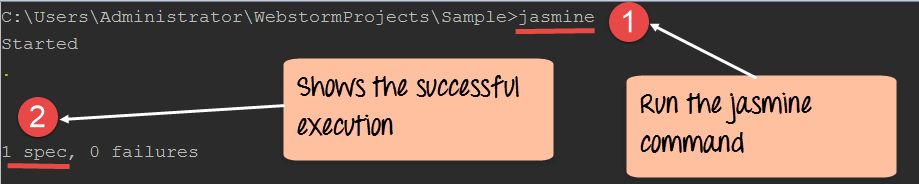
});

});

1. We need to first include our Add.js file so that we can test the 'AddNumber' function in this file.
2. We are now creating our test module. The first part of the test module is to describe a method which basically gives a name for our test. In this case, the name of our test is "Addition".
3. The next bit is to give a description for our test using the 'it' method.
4. We now invoke our Addnumber method and send in 2 parameters 5 and 6. This will be passed to our Addnumber method in the App.js file. The return value is then stored in a variable called value.
5. The final step is to do the comparison or our actual test. Since we expect the value returned by the Addnumber function to be 11, we define this using the method expect(value).toBe(the expected value).

**Output**

1. In order to run the test, one needs to run the command jasmine.
2. The below screenshot shows that after the jasmine command is run , it will detect that there is a test called add-spec.js and execute that test accordingly. If there are any errors in the test, it will be shown accordingly.

[](https://www.guru99.com/images/NodeJS/010716_0733_NodejsTesti4.png)

**Summary**

* In order to test a Node.js application, the jasmine framework needs to be installed first. This is done by using the Node package manager.
* The test code needs to be written in a separate file, and the word 'spec' should be appended to the file name. Only if this is done will jasmine be able to detect that a file needs to be run.
* To run the test, you need to execute the jasmine command. This will find all files which have the 'spec' word attached to it and run the file accordingly.

**Node.Js Vs AngularJS: Know the Difference**

**What is Node JS?**

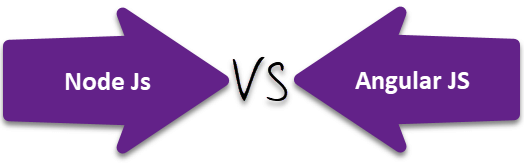
Node.js is a cross-platform runtime library and environment for running JavaScript applications outside the browser. This is a free and open source tool used for creating server-side JS applications.

Node.js applications are written in JavaScript. This application can runs within the Node.js runtime on Linux and Microsoft Windows. This framework offers a rich library of various JavaScript modules to simplify web development process.

**What is Angular JS?**

AngularJS is a structural framework for developing dynamic web apps. It allows developers to use HTML as template language and lets HTML's syntax to express application's components briefly but clearly.

It is a full-featured JavaScript framework, with the primary goal of simplification. It provides support for developing dynamic, single page web apps and supports the (MVC) programming structure.

[](https://www.guru99.com/images/1/120817_0700_NodeJsVsAng1.png)

**Node JS VS. Angular JS**

|  |  |
| --- | --- |
| **Node.JS** | **Angular JS** |
| Node.js is a cross-platform **run-time environment** for applications written in JavaScript language. | AnglarJS is an **open source** web application development framework developed by Google. |
| You **need to install Node.js** on the computer system. | You just need to **add AngularJS file** just like any other JavaScript file to use it in applications. |
| It is **written in C, C++ and JavaScript** languages. Node.js supports non-blocking input/output API and an event-driven architecture. | AngularJS is written **entirely in JavaScript**. It also allows extending the HTML's syntax to describe the components of your application. |
| Node.js is useful to build f**ast and scalable** server-side networking applications. | This framework is best suited for building **single-page** client-side web applications. |
| Node.js has many **different frameworks**like Sails.js, Partial.js, and Express.js,. | AngularJS is itself **web application framework**. |
| It is Ideal for developing **small size**projects. | It is Ideal for creating **highly active and interactive** web apps. |
| Developers can use JavaScript on the client as well as the server. So they can focus on **learning one language**. | Angular JS needs a deep understanding of **prototyping, scope,**and various other JavaScript aspects. |
| Node.js is quite **useful for generating database queries** because JavaScript is used to write database queries for databases like MongoDB, CouchDB. This reduces the developer's task as they don't need to remember the syntax differences while combining Node.js and NoSQL databases. | Almost all other frameworks implement MVC by asking **splitting an app into MVC components**. It is difficult task, but Angular makes it simple as it implements MVC by splitting the app into MVC components. |
| Node.js will be an ideal option for the **real-time collaborative drawing/editing-type apps**, where the user can see the document being modify live by another user like Dropbox or Google Docs. | Angular's declarative approach simplifies app development in a sort of **WYSIWYG**way. Instead of spending time on how the program flows and what loaded first the developers can define what they want, and Angular JS will take care of the dependencies. |
| Node.js will be useful in situations when something **faster and more scalable** is needed. | Angular is useful in creating **real time applications** like instant messaging or chat apps. |

**What Is Better Node JS Or Angular JS?**

Both Node.js and AngularJS platforms are targeted at building web applications using JavaScript. However, both of them are different in their architecture and working. Node.js mainly used to build server-side applications. AngularJS is suited for building single-page client-side web applications.

These two frameworks are high performing, advanced and widely used worldwide. So one should select according to his/her project need.

**Conclusion:**

* Node.js is a cross-platform runtime library and environment for running JavaScript applications outside the browser.
* Node.js offers a rich library of various JavaScript modules which can simplify coding.
* AngularJS is a structural framework for developing dynamic web apps.
* The goal of Angular JS framework is to provide simplified the development process.
* Node JS is Ideal for developing small size projects while Angular JS is a great option for long size projects.
* Both Node.js and AngularJS platforms are targeted at building web applications using JavaScript. However, both of them are different in their architecture and working

# Node.js Vs Python: What's the Difference?

## What is Node.js?

Node.js is a server-side platform built on Google Chrome's JavaScript Engine. It uses a non-blocking, event-driven I/O model. It allows developers to create data-intensive real-time applications that run across distributed devices. Its applications are written in JavaScript. It can be run on OS X, Microsoft Windows, and Linux operating systems. It is widely used to run real-time server applications.

## What is Python?

Python is an object-oriented, high level, dynamic and multipurpose programming language. Python's syntax and dynamic typing with interpreted nature, make it an ideal language for scripting.

It supports multiple programming patterns, including object-oriented programming, functional programming, or procedural styles. Moreover, it an interpreted language which means it cannot convert to computer-readable code before its runs at runtime.

In this compariosn, we will cover

* [Features of Node js](https://www.guru99.com/node-js-vs-python.html#1)
* [Features of Python](https://www.guru99.com/node-js-vs-python.html#2)
* [Node JS Vs. Python](https://www.guru99.com/node-js-vs-python.html#3)
* [When to use Node js?](https://www.guru99.com/node-js-vs-python.html#4)
* [When to use Python?](https://www.guru99.com/node-js-vs-python.html#5)

### Features of Node js

* It uses a single threaded model with event looping. This type of event mechanism benefits the server to respond in a non-blocking way.
* It is built on V8 JavaScript Engine makes it fastest code execution library.
* There is no buffering in Node.js as applications output the data in pieces.

### Features of Python

* It allows low-level modules inclusion to the Python interpreter. These modules allow programmers to add or customize their tools.
* It provides interfaces to all major commercial databases.
* It supports functional and structured programming methods as well as OOP.
* It offers high-level dynamic data types and supports dynamic type checking.
* It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

[](https://www.guru99.com/images/1/111517_1156_NodejsvsPyt1.png)

## Node.JS Vs. Python

|  |  |
| --- | --- |
| **Node.JS** | **Python** |
| Node.js is best suited for asynchronous programming | Python is **not** exactly the best option for asynchronous programming. |
| Node.js is pure JavaScript, so its basics remain simple for the developers to learn. | The biggest advantage of using Python is that developers need to write fewer lines of code. |
| Node.js lacks the clean coding standards. That's why it cannot be recommended for larger projects. | It is ideal for a large project as it can do anything that can be done using PHP code. |
| Node.js is an ideal platform available right now to deal with real-time web applications. | Not an ideal platform to deal with real-time web applications. |
| It best suited for small projects to enable functionality which needs less amount of scripting. | Python is suited for developing larger projects. |
| Best for memory intensive activities. | Not advisable for memory intensive activities. |
| Node.js is a better choice if your focus is on web applications and website development. | Python is an ideal platform to do multiple things - web applications, integration with back-end applications, numerical computations, machine learning, and network programming. |
| Node.js utilizes JavaScript interpreter. | Python using PyPy as Interpreter. |
| Node.js supports callback. Its programming is based on event/callback that makes it process Faster. | It supports generators which makes it much simpler. |

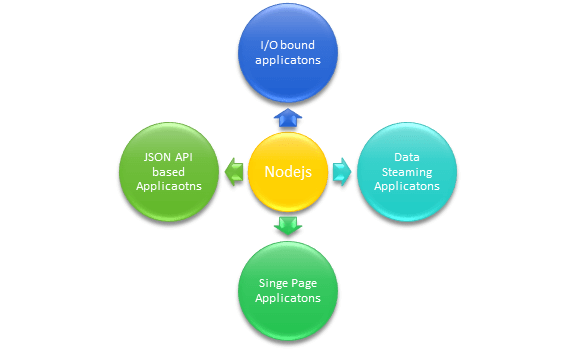
Selecting a development platform is significant steps in the app development phase. Node.JS and Python are certainly two highly discussed languages nowadays. Let where we can use Node JS and when to use python.

## When to use Node js?

As Node.js is a platform built on Chrome's JavaScript runtime it helps to develop building scalable network applications. It uses an event-driven, non-blocking I/O model which makes it an ideal option for developing for data-intensive real-time applications.

Node.js offers greater performance and speed. It is an ideal solution for developing messaging or chatting applications. It is also useful for developing heavy-load applications and e-commerce sites that depend on the speed of processing.

**Apps best suited with Node JS:**

[](https://www.guru99.com/images/1/111517_1156_NodejsvsPyt2.png)

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

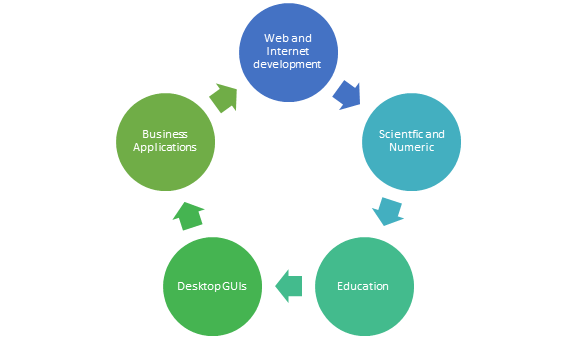
## When to use Python?

Python has a library of open source data analysis tools, web frameworks, and testing instruments. Therefore, its ecosystem one of the largest out of any programming community.

Python is a most accessible language for new programmers as community provides many introductory resources. The language is also taught in universities and used for working with beginner-friendly devices such as the Raspberry Pi.

Python is also featuring consistency, stability, and easiness of use. It is highly preferred for developing scientific applications, big data solutions, and government projects.

**Apps best suited with Python**

[](https://www.guru99.com/images/1/111517_1156_NodejsvsPyt3.png)

## KEY DIFFERENCE

* Node.Js is a server-side platform built on Google Chrome Javascript Engine whereas Python is an object-oriented, high level, dynamic and multipurpose programming language.
* Node is better for web applications and website development whereas Python is best suitable for back-end applications, numerical computations and machine learning.
* Nodejs utilize javascript interpreter whereas Python uses PyPy as an interpreter.
* Node is best suited for asynchronous programming whereas Python is not the best option for asynchronous programming.
* Node.Js is best suited for small projects while Python is best suited for developing large projects.
* Node is best for memory-intensive activities whereas Python is not advisable for memory-intensive activities.