**What is Node.js?**

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

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

## Why use Node.js?

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

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

var http = require('http');  
  
http.createServer(function (req, res) {

  res.writeHead(200, {'Content-Type': 'text/plain'});  
  res.end(Test Server');

}).listen(8081);

**To run the node code**

$node index.js

## Include Modules

To include a module, use the require() function with the name of the module:

var http = require('http');

Now your application has access to the HTTP module, and is able to create a server:

http.createServer(function (req, res) {  
  res.writeHead(200, {'Content-Type': 'text/html'});  
  res.end('Test Server');  
}).listen(8081);

## Features of Node.js

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

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

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

var fs = require('fs');

fs.readFile("Sample.txt",function(error,data)

{

console.log("Reading Data completed");

});

* The above code snippet looks at reading a file called Sample.txt. In other programming languages, the next line of processing would only happen once the entire file is read.
* But in the case of Node.js the important fraction of code to notice is the declaration of the function ('function(error,data)'). This is known as a callback function.
* So what happens here is that the file reading operation will start in the background. And other processing can happen simultaneously while the file is being read. Once the file read operation is completed, this anonymous function will be called and the text "Reading Data completed" will be written to the console log.

1. Node uses the V8 JavaScript Runtime engine, the one which is used by Google Chrome. Node has a wrapper over the JavaScript engine which makes the runtime engine much faster and hence processing of requests within Node also become faster.
2. Handling of concurrent requests – Another key functionality of Node is the ability to handle concurrent connections with a very minimal overhead on a single process.
3. The Node.js library used JavaScript – This is another important aspect of development in Node.js. A major part of the development community are already well versed in javascript, and hence, development in Node.js becomes easier for a developer who knows javascript.
4. There are an Active and vibrant community for the Node.js framework. Because of the active community, there are always keys updates made available to the framework. This helps to keep the framework always up-to-date with the latest trends in web development.

## Who uses Node.js

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

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

## When to Use Node.js

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

1. Chat applications
2. Game servers – Fast and high-performance servers that need to processes thousands of requests at a time, then this is an ideal framework.
3. Good for collaborative environment – This is good for environments which manage document. In document management environment you will have multiple people who post their documents and do constant changes by checking out and checking in documents. So Node.js is good for these environments because the event loop in Node.js can be triggered whenever documents are changed in a document managed environment.
4. Advertisement servers – Again here you could have thousands of request to pull advertisements from the central server and Node.js can be an ideal framework to handle this.
5. Streaming servers – Another ideal scenario to use Node is for multimedia streaming servers wherein clients have request's to pull different multimedia contents from this server.

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

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

## When to not use Node.js

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

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

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

Once you have downloaded and installed Node.js on your computer, lets 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 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 7000. 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

**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 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 for node and browsers.
4. **MongoDB** - The MongoDB 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 focus on innovative features and performance

**Using modules in Node.js**

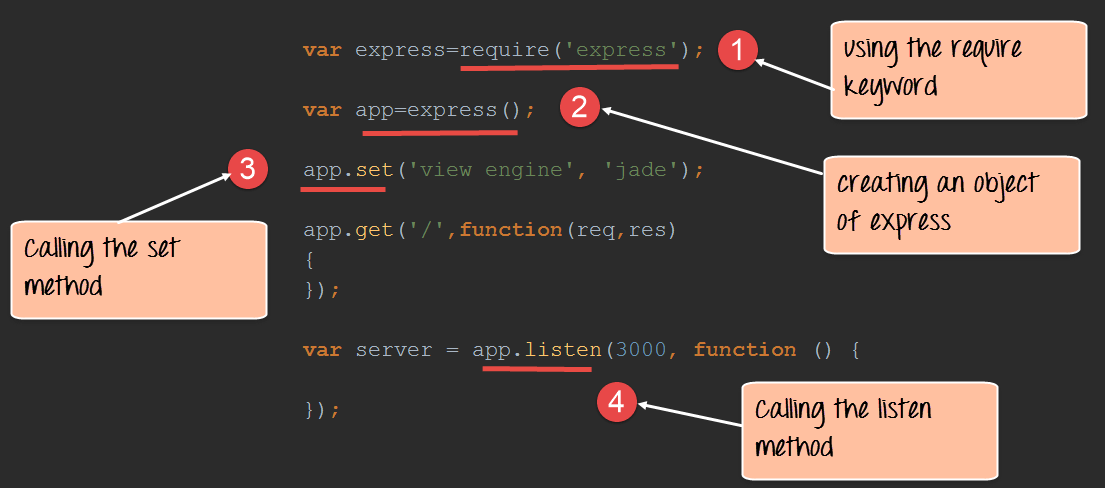
In order to use modules in a Node.js application, they first need to be installed using the Node package manager.

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

**npm install express**

* 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 how we can use the "require" keyword. The below "Vision" code example shows how to use the require function



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 method to make the application listen on a particular port number.

**Creating NPM modules**

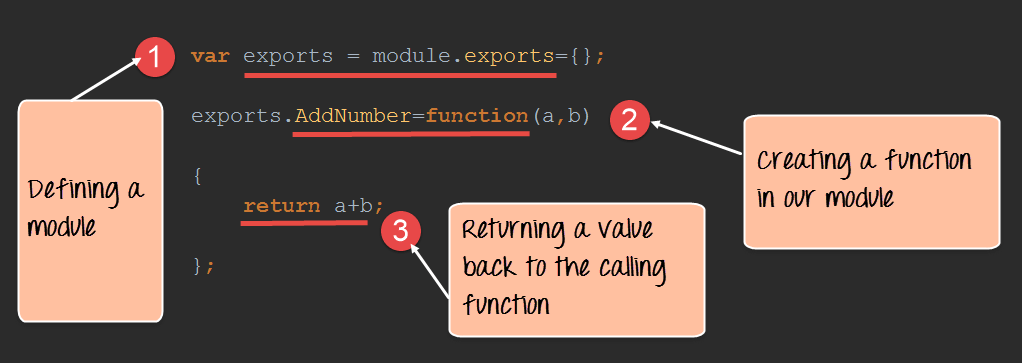
Node.js has the ability to create custom modules and allows you to include those custom modules in your Node.js application.

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 2 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;



var exports=module.exports={};

exports.AddNumber=function(a,b)

{

return a+b;

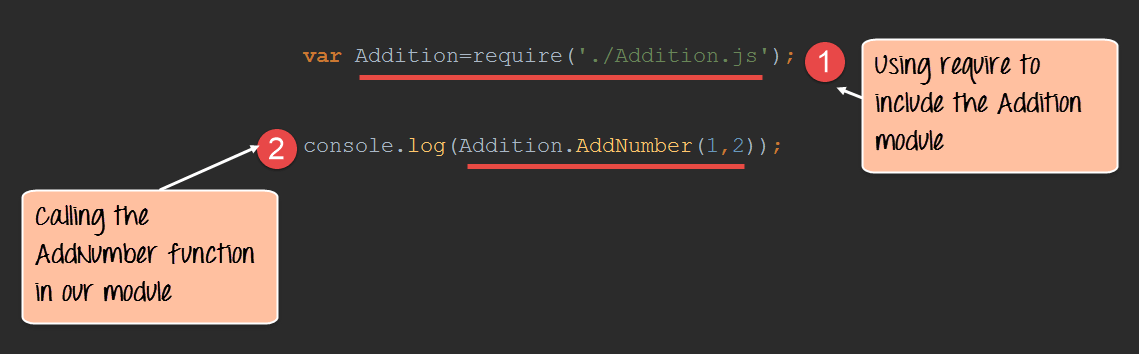
};

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.

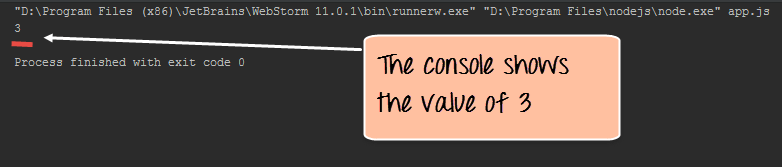
**Step 2)** Create a file called "app.js," which is your main application file and add the below code



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.



**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.

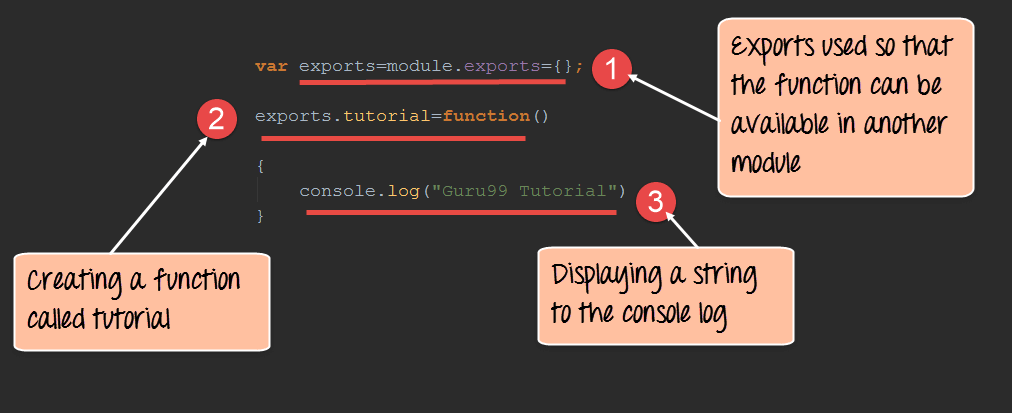
In modern day programming, it's quite common to build a library of common modules and then extend the functionality of these common modules if required.

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 "Vision Tutorial".



var exports=module.exports={};

exports.tutorial=function()

{

console.log("Vision Tutorial")

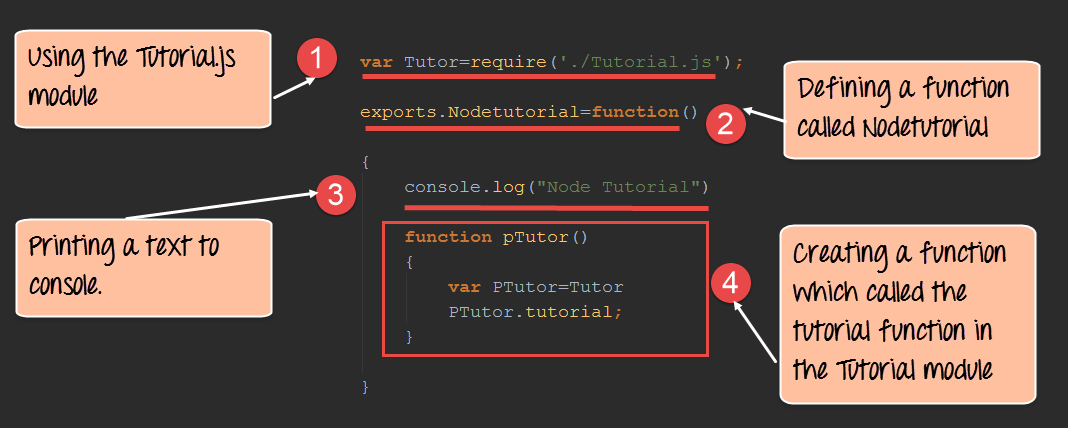
}

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 "Vision 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.



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();

}

}

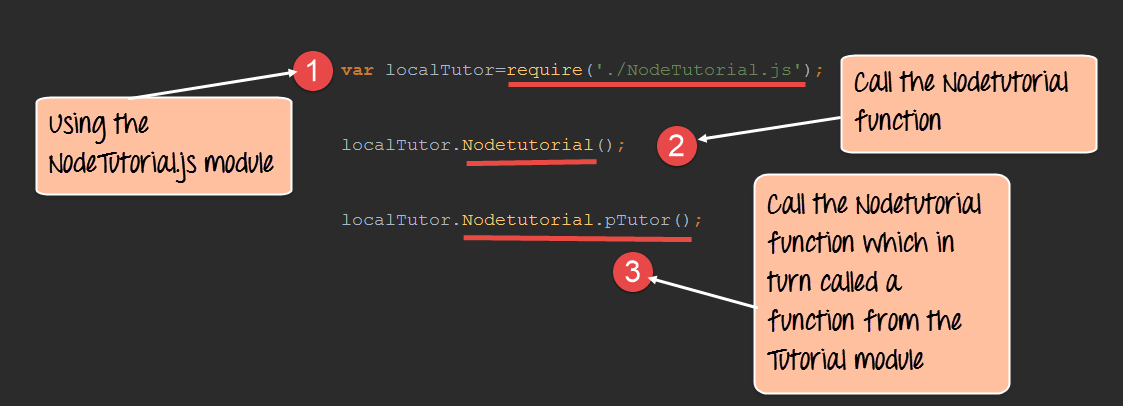
Note, the following key points about the above code

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 "Vision 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 "Vision Tutorial" to the console.log.

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



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 "Vision 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
* Vision 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 "Vision."

npm set init.author.email "Vision@gmail.com"

npm set init.author.url http://Vision.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**

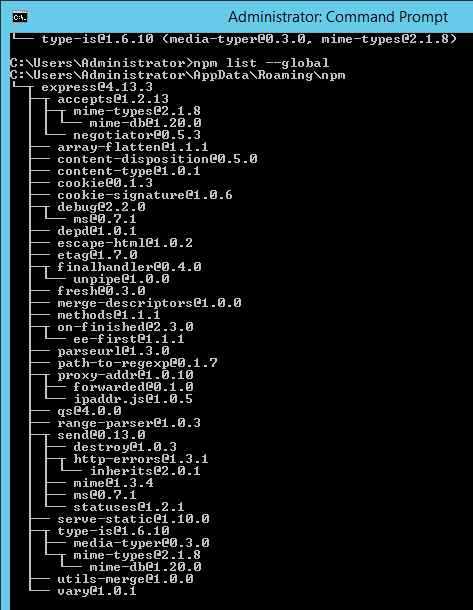
The global option in the above statement is what allows the modules to be installed at a global level.

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.



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 what is the package 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. Un-installing a package – The same in which you can install a package, you can also un-install 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**

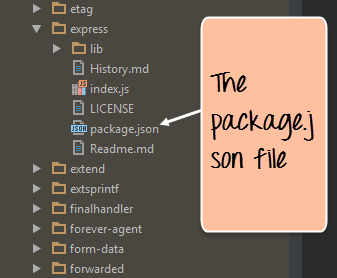
The "package.json" file is used to hold the **metadata about a particular project**. This information provides the Node package manager the necessary information to understand how the project should be handled along with its dependencies.

The package.json files contains 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.

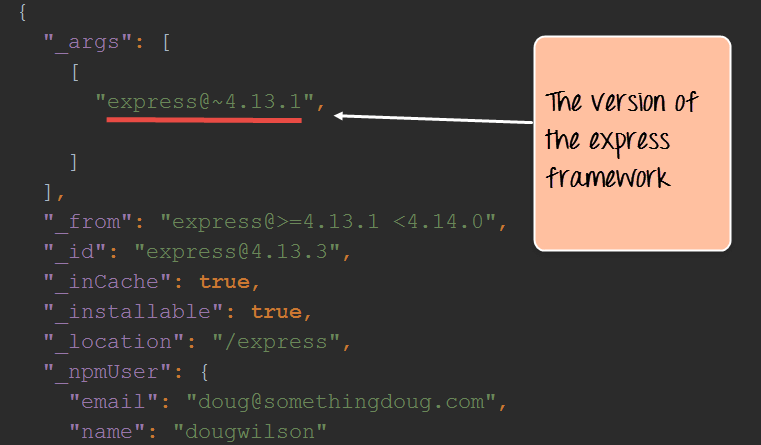
Let's take an example of how the structure of a module looks when it is installed via npm.

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.



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.



**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.