ATTUNE

Getting Started with NodeJS.

By Juned Laliwala

Short description of NodeJS version-5.9.0

- Installation on Windows
- Working with NPM.

ATTUNE

Contents

1. Preface

1.1. About This Guide

NodeJS is a powerful Javascript which is being built on Google Chrome's Javascript Engine. So the main intense of this guide is to cover up the basic ideas about the node to the readers. The readers at the end of this guide will definitely being able to make different and innovative projects. Nowadays technology is increasing day by day and as a result more powerful script is needed, so we are here to make and understand different aspects of nodeJS. We will also come across various features of nodeJS including the usage area of the framework.

1.2. Intended Audience

This guide is basically for all the students who are currently just introduced into the IT environment. This is also for the developers of various languages which are in current demand. With the help of this guide all will be able to make various web sites and also subsequently various applications.

1.3. Revision History

This is the primary version of NodeJs. This version will cover up installation, features, advantages, disadvantages and also various functionalities of the nodeJS.

2. Introduction to NodeJS

Node.js is a powerful framework which is based on JavaScript. This was built by Google Chrome's Javascript v8 Engine. With the help of node.js we can make various applications including video streaming websites, Single-Page Applications (SPA), and also other Web Applications. Node.js is completely available for free and it is being used by thousands of developers all around the world.
The JavaScript is becoming MUST for all the developers nowadays. Front-End developers use this script for the various purposes like adding user interface enhancements, adding interactivity and talking back to back for various interactions with the clients.
Whenever you install node.js into your system, you will come across against one of the most important part of this framework that is "Node Package Manager (NPM)". NPM is nothing but the tools to install various packages that are being needed into our project. We will study more about npm in the future sections of this guide.

2.1 Features of Node.js

- **Asynchronous and Event Driven**: Non-blocking of the data occurs when we are referring to node.js framework. In other words, we can say that the server which has been made by node.js will never wait for an API to return the data. This asynchronous way is being handled by the events subsequently and accordingly returns the event that the intended work is done.
- **Speed is Fast**: Node.js which is built up by Chrome's v8 engine is very super fast.
- **Single Threaded but Highly Scalable**: Node.js uses a single threaded program and the same program can provide service to a much larger number of requests than traditional servers like Apache HTTP Server.
- **License**: Node.js is released under the MIT license

2.2 Where to Use Framework?

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

2.3 Where not to use??

. CPU Intensive Applications

2.4. Installation on Windows OS.

Nodejs can be easily downloaded from the official website of nodejs (nodejs.org). Download the nodejs which is particularly for the windows that we are using. When you install the software you will see the "Custom Setup" in which npm will be installed along with node.js.

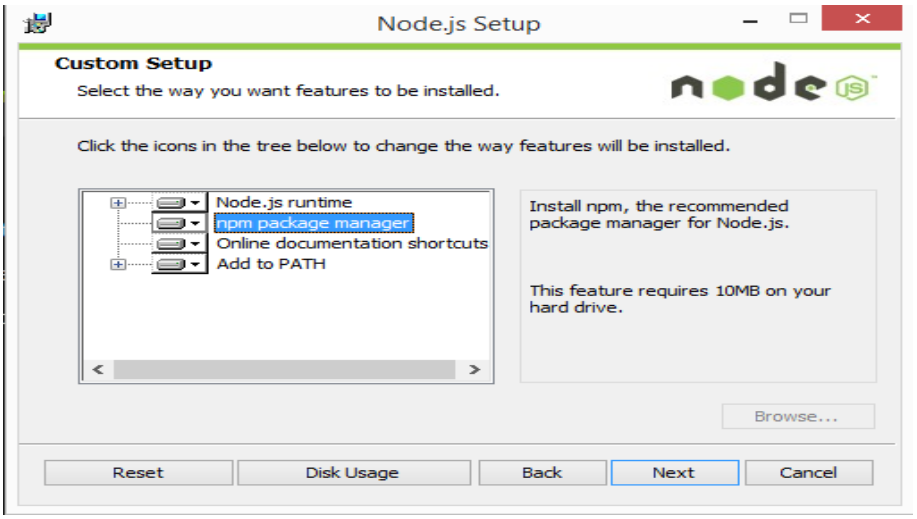

Figure 1 Node.js Setup

2.5 Testing a Node

This section deals with the testing a node after node.js has been successfully installed into your machine. Testing is generally to check for the version of the node and npm that we have installed.

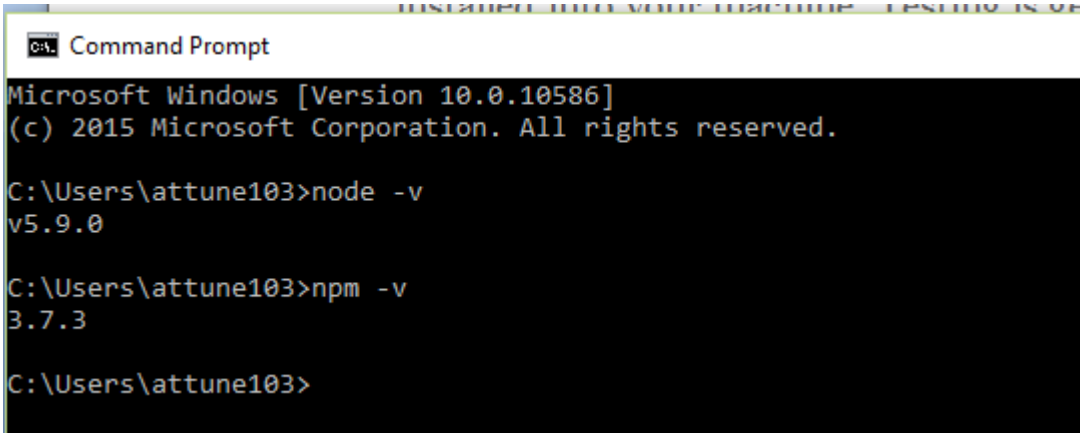

Figure 2 node -v and npm -v

3. Basic Concepts

This section will include the file structure of the node.js framework. Along with file structure you will come across the basic examples for running the node.js programs.

3.1. First Example

To start with our basic example, we are making a folder named “AttuneInfo” into our root directory. Inside the AttuneInfo, we are making our new file named “app.js”. This app.js is nothing but the javascript file that will be running through node.js.

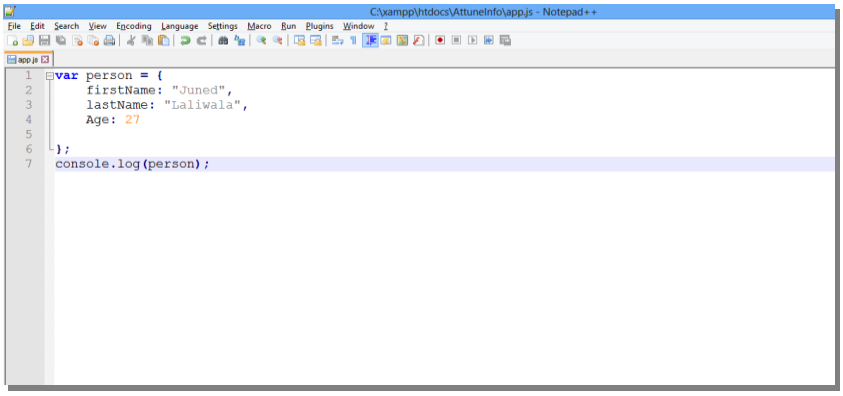

Figure 3 First Example

As you can see from the above figure that we have included one variable “person” person which are holding the three parameters namely firstname, lastname and age.

Console.log () is the function through which we will see the message in our command prompt whenever our program is running.

Now we will run the program:

To run a program the following command is required: node app.js

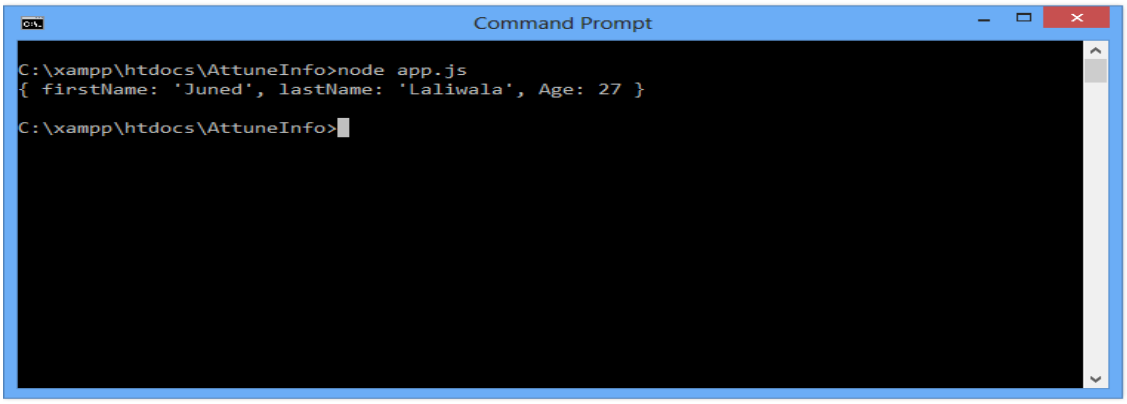

Figure 4 Output of the first example

As we can see from the above example that we have made one javascript file in which we have included a person variable with some parameters. After applying some parameters we have run the program using "node" and the parameters that we have passed are easily seen in our specific output.

3.2 Second Example

In the first example we have seen the way to display the person. Now in the second example we are going to add two numbers using function.

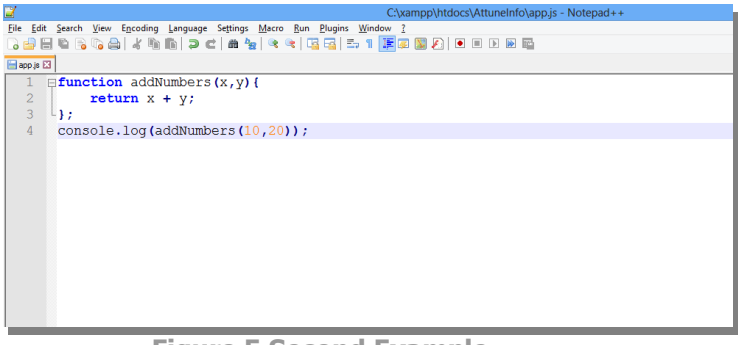

Figure 5 Second Example

As you can seen from the fig:6 that we have made a function named "addNumbers". With the help of this function we are passing two values inside the function. The addition of this two numbers takes place and then we run the program using node.

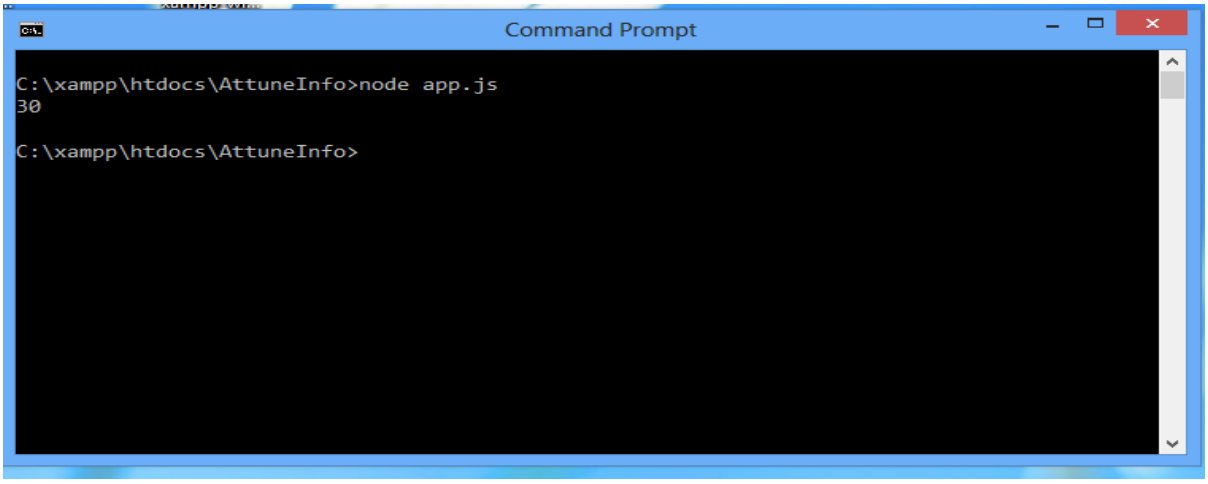

Figure 6 Output of Second Example

3.3. Third Example

In this example we will not pass any parameters with our function. So let us see what's the output. The function is having nothing in it so we have named a function as "Worthless".

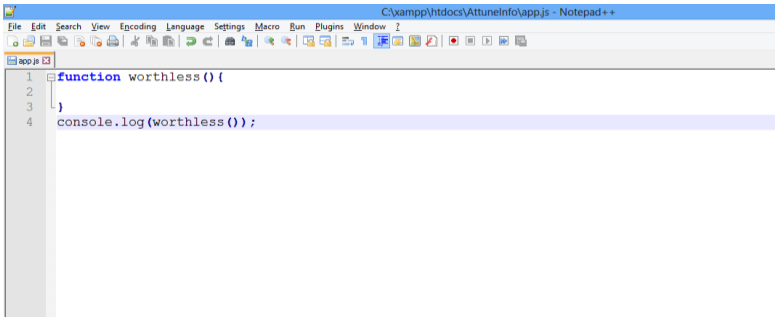

Figure 7 Third Example

Here we have defined one function named "worthless" with not a single parameter. This function is totally blank. So the desired output is as follows:

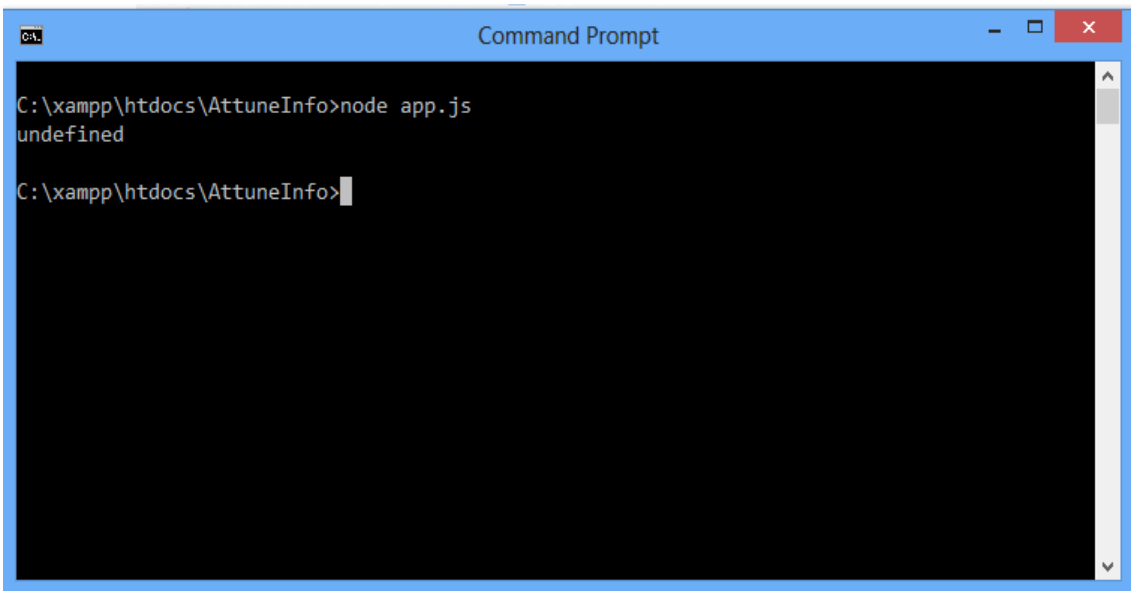

Figure 8 Output of Third Example

The output is “undefined” which means that whenever the compiler does not find anything inside the function it will return “undefined”.

3.4. Fourth Example

Here we will make one variable as a function and inside that function we will pass some value. After passing some value we will at the end calls that function.

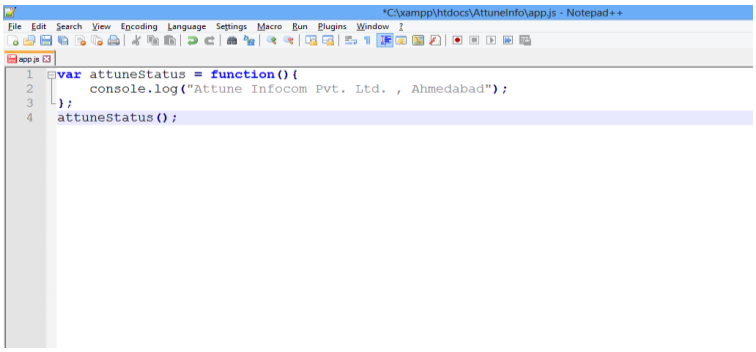

Figure 9 Fourth Example

In the above diagram we can see that we have made one variable named “attuneStatus” and we have made it a function with a parameter passing the information. After that, we have called that function (attuneStatus( )).

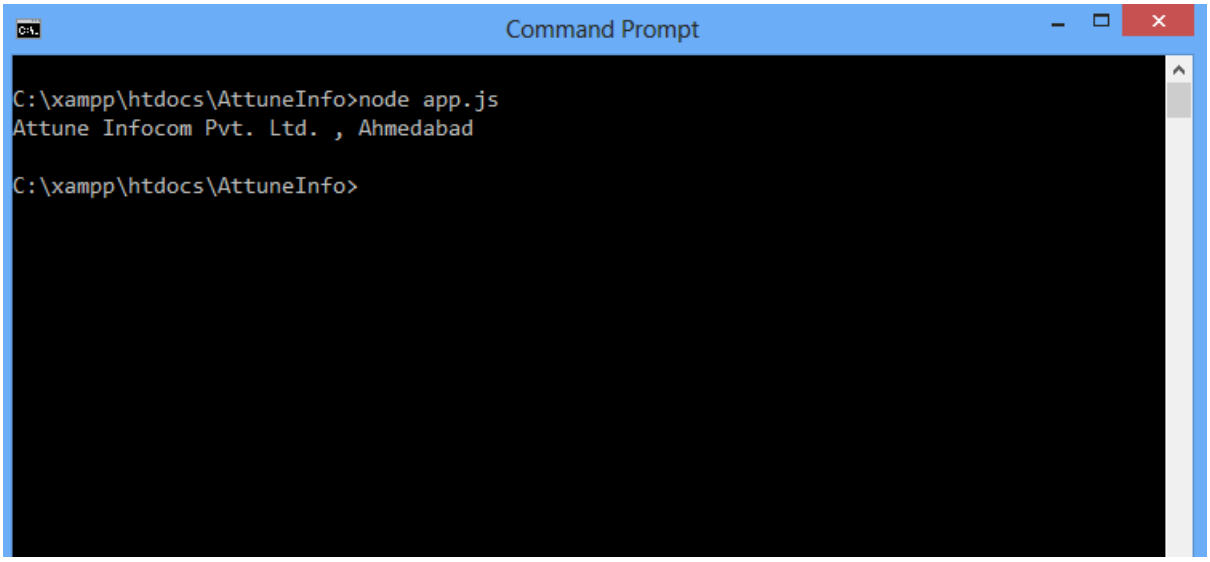

Figure 10 Output of Fourth Example

In our example we are adding one more functionality. This functionality is to setTimeOut() which means that after a periodic interval of 5 seconds output will be regenerated as follows:

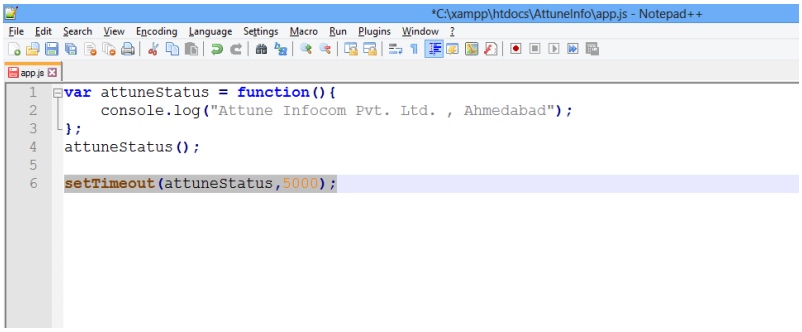

Figure 11 Fifth Example

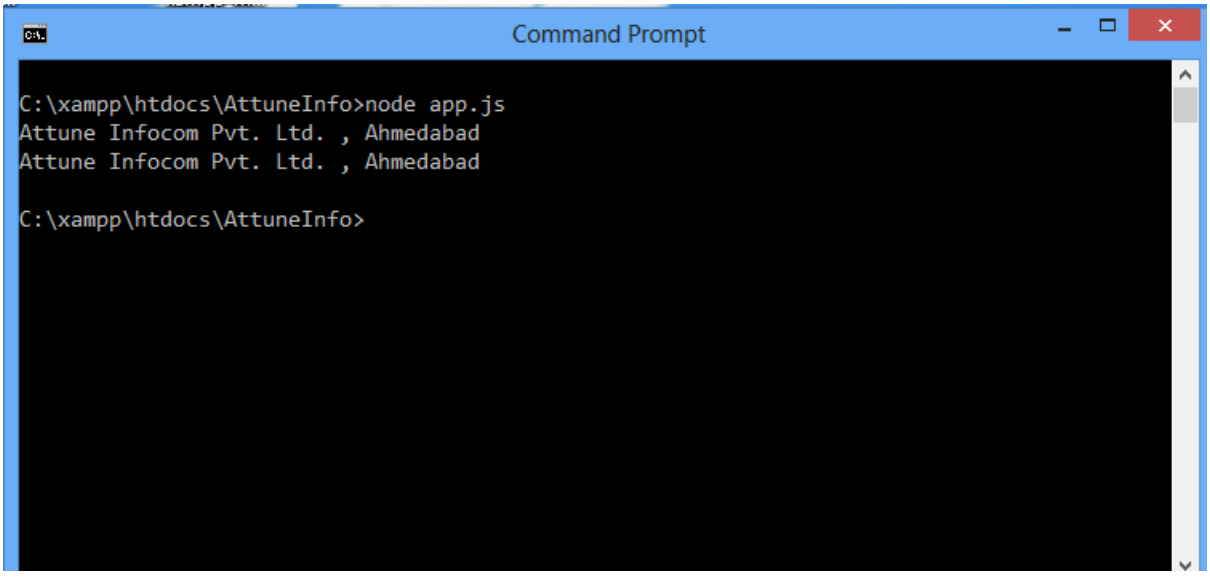

Figure 12 Output fifth Example

So the desired output is above. The second line of the output will be displayed successfully after a period of about 5 seconds.
So these were the basic concepts of nodeJS. Now we will move forward to handle the request from the server.

4. Handling Multiple Request

Handling multiple request means the request which is being made by one function is being used by another function then we can say it as multiple handling request. In a simple term we have made a good example to review the request. We will explain this with a function named “placeanEmployee” with a parameter “TechName”.

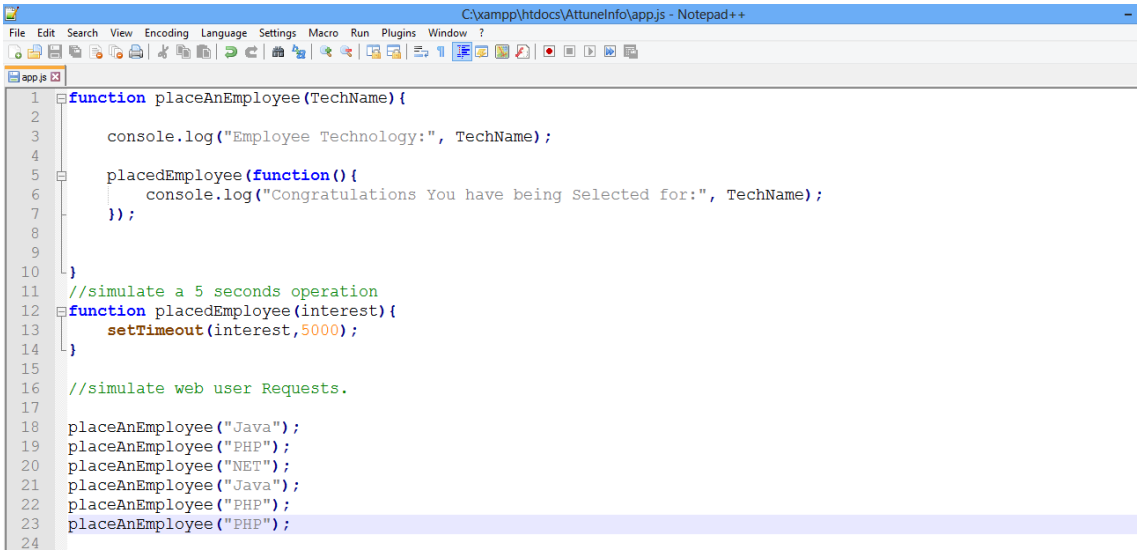

Figure 13 Handling Multiple Request

Looking at the above example it seems tedious. But Hold on. We will make it simple. We have made one function named “placeanEmployee” in which this function will pass one parameter named “TechName” which is nothing but the name of the technology in which he is interested.

In the next line we are just passing the information in which the employee is interested. We have defined one more function named “placedEmployee” which will place the employee in his own interested area, so we have passed a parameter named “interest”. Now we are calling “placedEmployee” inside the “placeanEmployee”.

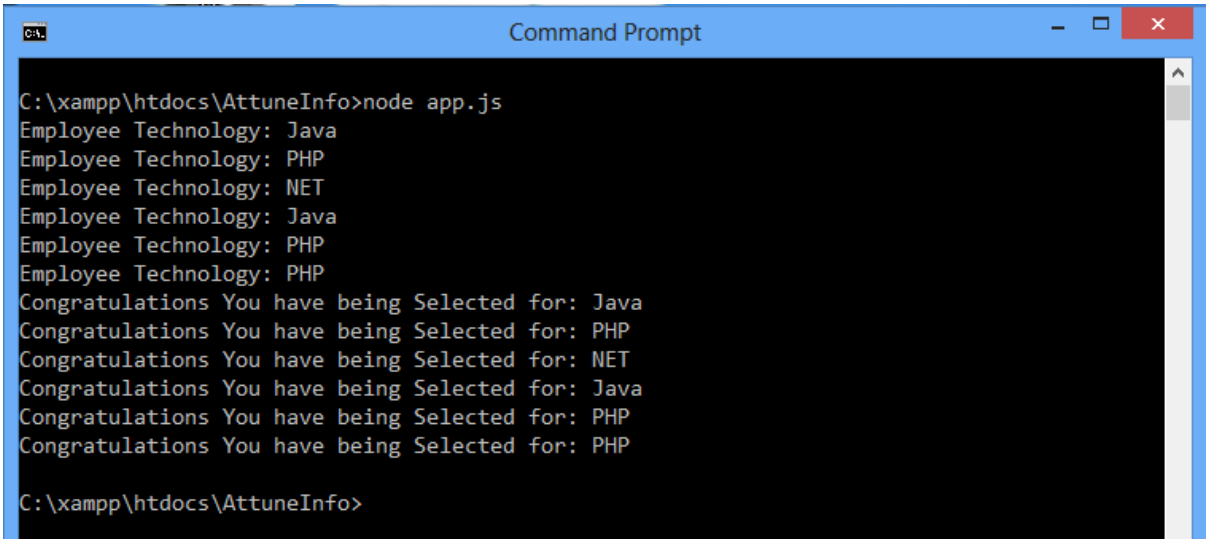

Figure 14 Output Handling Multiple Request

As we can see from the output that we have passed different values in placeanEmployee( ) function and as a result we can see that all the values have been successfully posted to the employee list as defined in placedEmployee().

5. Understanding References to Objects

Reference to object is in simple term is to link one function or variable to another. So in our examples also we are going to reference one object to another. This object can be either function, variable, files and so on.

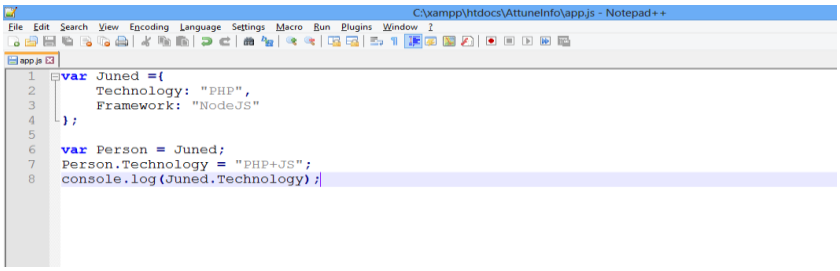

Figure 15 References to objects

From the above diagram we can see that there is a variable named “Juned” and inside it there are two parameters namely “Technology” and “Framework”. Now this variable “Juned” is being now referenced to another variable named “Person”. So now “Person” will now handle all the values that “Juned” was holding.

To make it clear, we have changed the parameter of “Technology” in “Person” and in the next line we are consoling “Technology of “Juned”.

The Output of the above example is as follows:

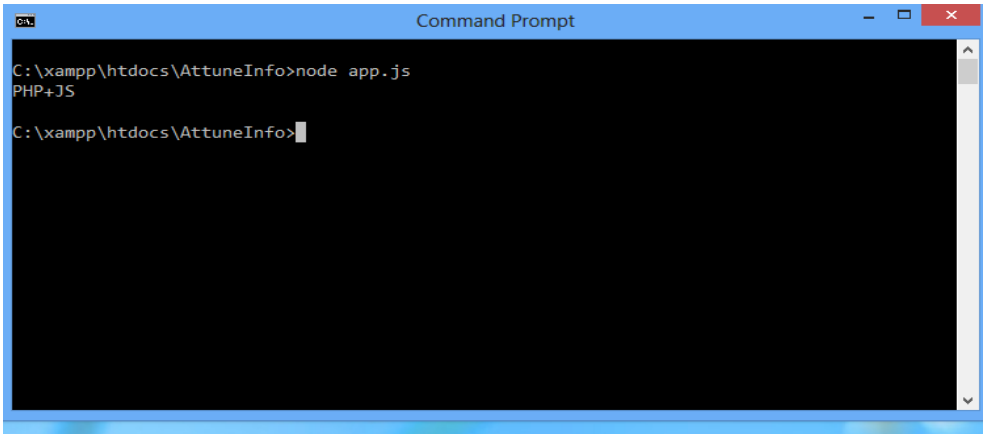

Figure 16 Output References to Objects

6. This

In JavaScript, the thing called **this**, is the object that "owns" the JavaScript code. The value of **this**, when used in a function, is the object that "owns" the function. The value of **this**, when used in an object, is the object itself.

The **this** keyword in an object constructor does not have a value. It is only a substitute for the new object. The value of **this** will become the new object when the constructor is used to create an object.

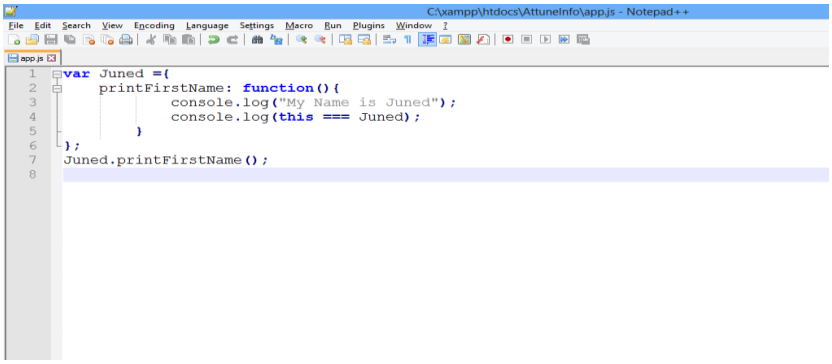

Figure 17 This

In the above diagram we can see that inside the Juned variable we have made a function named “printFirstName”.
Inside that we have made one statement.
Now we are passing “this” parameter to check the value. Remember that whatever value you pass in this parameter will be checked against the output.
To make it clear see the last line “Juned.printFirstName();” so this Juned will be checked and the desired output will be:
With the help of this keyword, we can give reference to same function or variable.

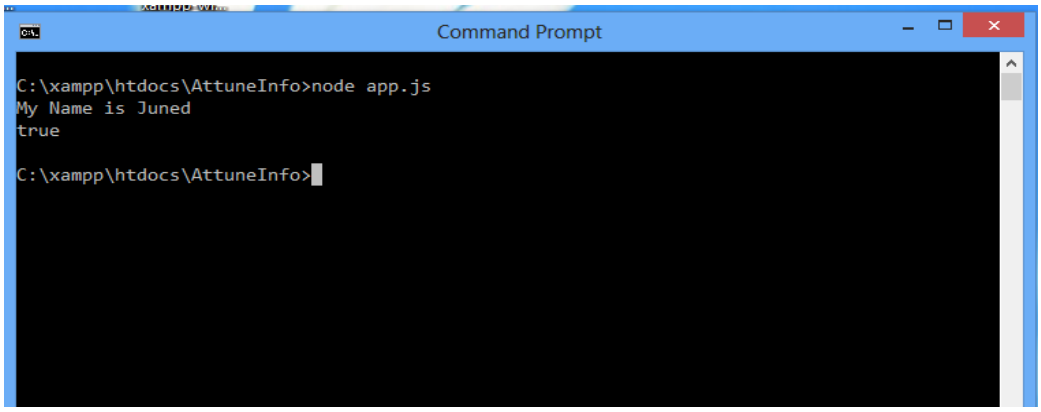

Figure 18 Output This

So first line is nothing but the console.log information
The second line interprets this as “TRUE” because the value “Juned” is same in both the cases.

7. Prototype

Before we start to understand the concept of prototype, let us make one example to demonstrate the use of prototype. We are making one user function and inside that function we are passing two parameters namely “this.name” and “targetplayer.name”. This example is regarding giving age life to another person.

7.1 Example without Prototype

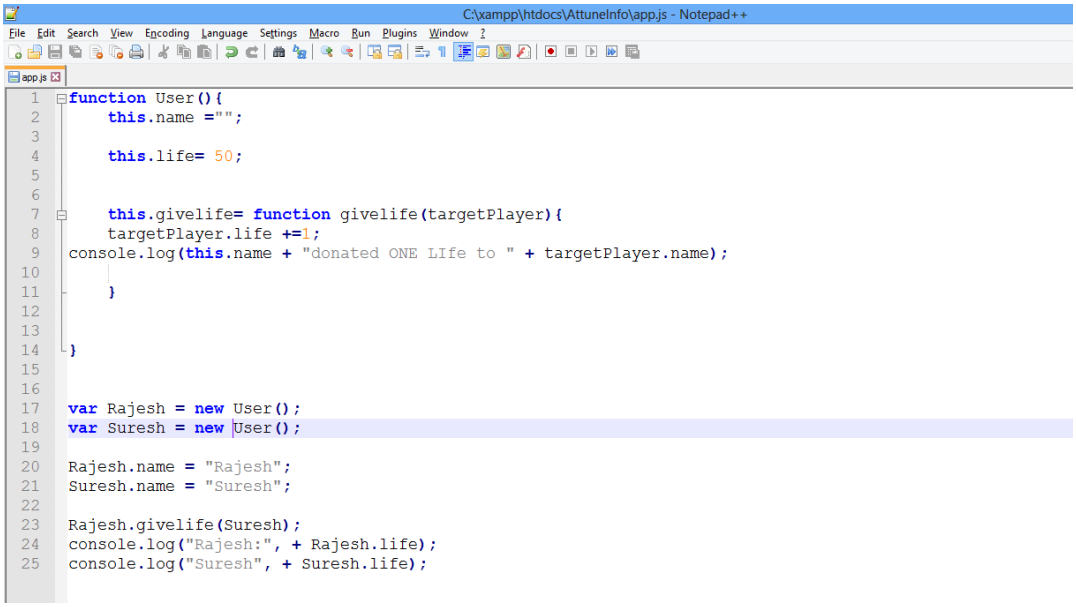

Figure 19 Example without Prototype

From the above diagram we can see that the current user has been assigned a value of life to be 50, and in the next line we can see that the current user is giving his life to another user. This value is incremented by one. So last we are passing the console information that the current user has given his one life to another user.

The Current user is Rajesh and the target user is Suresh.

Below diagram shows the output of the above example. Rajesh life is 50 and Suresh life is 51.

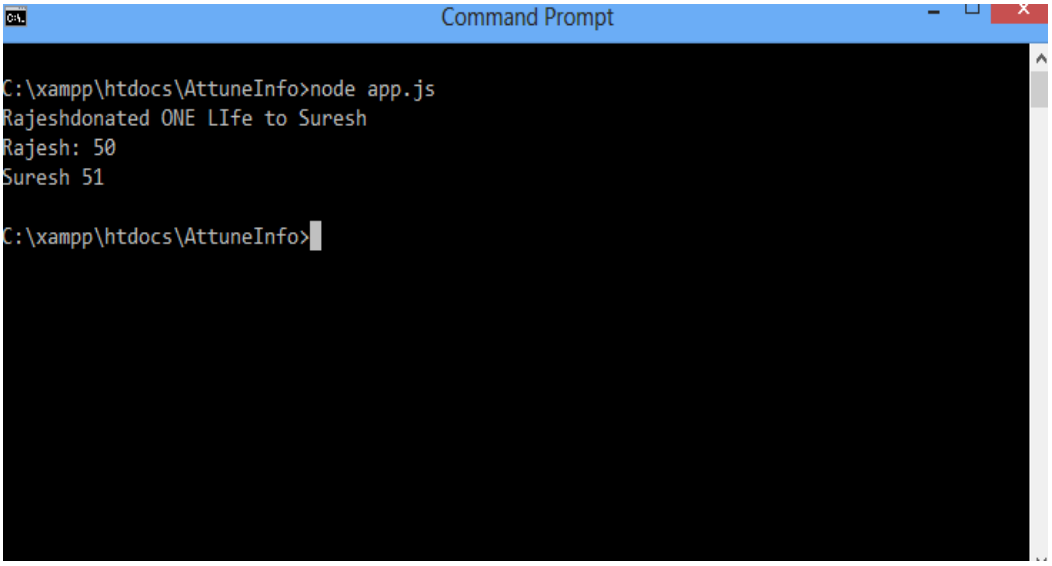

Figure 20 Output without Prototype

7.2 Example with Prototype

Now we will see the definition of prototype and subsequently we will demonstrate the example of prototype.

PROTOTYPE: Prototype is nothing but an object. All the JavaScript objects are inheriting their properties and method from the prototypes.

Now in our case we are making one prototype named “cutter” which is reducing the life of the current user as shown in the figure below:

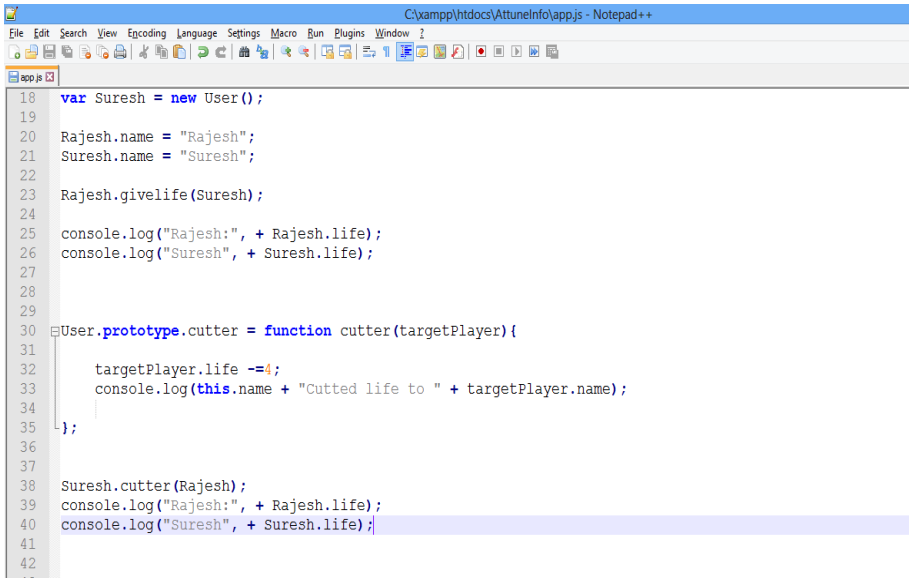

Figure 21 Example of Prototype

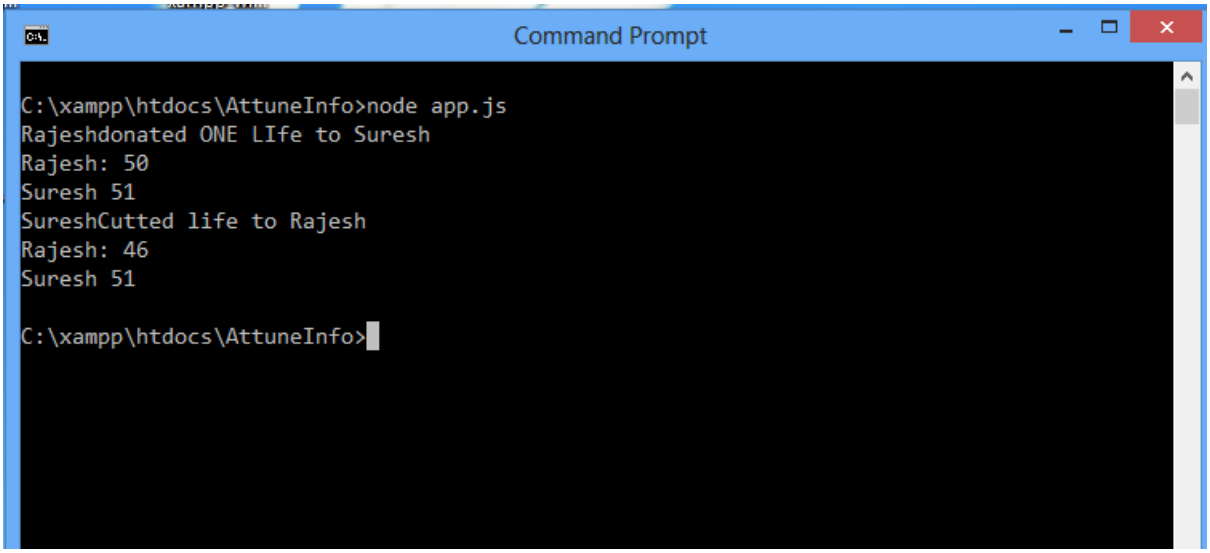

Figure 22 Output with Prototype

7.3 One more Prototype.

Here we will see one more example of prototype. In this prototype, we will add a specific value to both the users and will display it in the output.

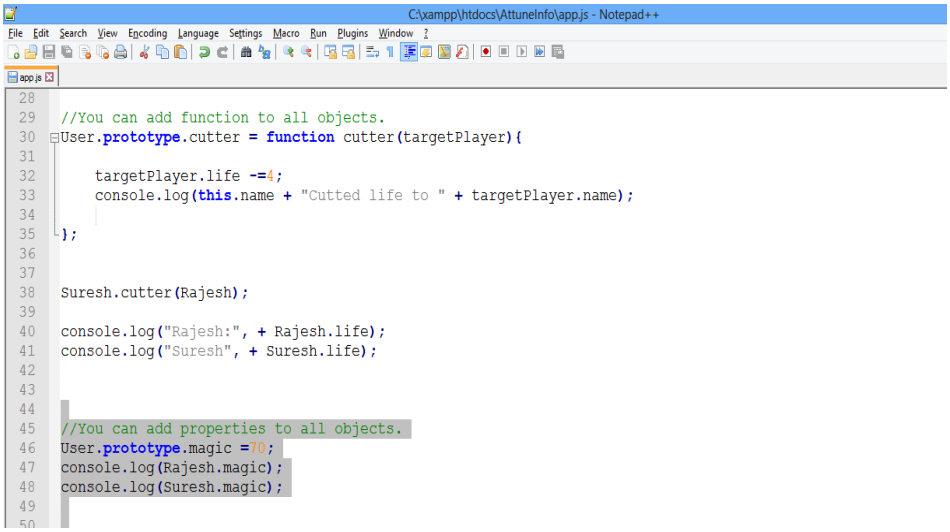

Figure 23 One More Prototype

In the above diagram, we can see that we have made one prototype named “magic”. Inside this magic we are assigning the default value to 70, so when we run the above example the output will be as follows:

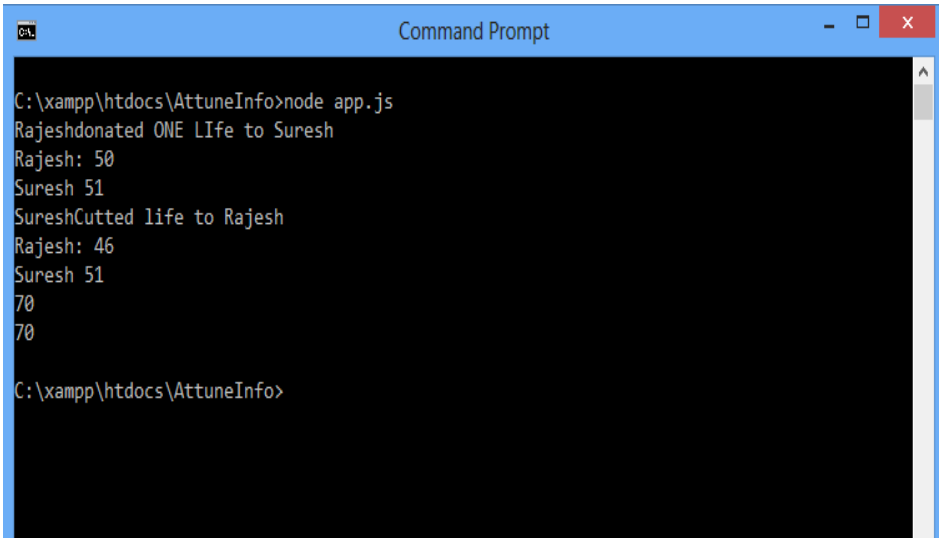

Figure 24 Output One more prototype

8. Modules

Modules are the core part of the node.js framework. In simple language we can say that files and modules are in one-to-one correspondence. When one file is going to use another file or function then we can say that we have to make a module.
This important part can be easily understood by an example.
In our example, we are now working with the “employee.js”. This file will basically handle two main functions namely “AddEmployee” and “LeaveEmployee”. Inside this both function we are passing some information to check the functionality of both the functions.

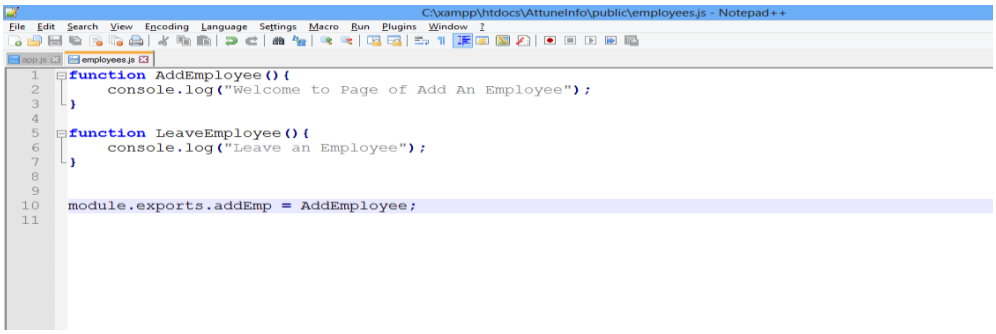

Figure 25 Employee Module

Now the main part is that we have to bind these modules into our main “app.js” file. This app.js file will now be solely responsible for handling all the activities of the employees.

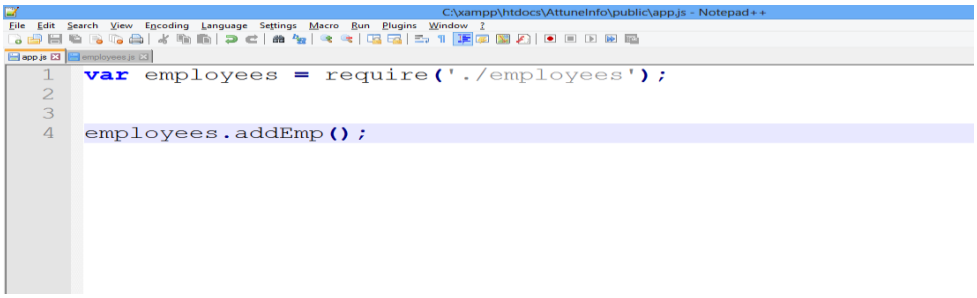

Figure 26 app.js in Module

From the figure 26 we can say that we have made a variable named “employees” and we are linking that variable to our employees.js file using require parameter. Now we are calling addEmp() function which is in turn responsible for adding the employees details as stated in figure 25.
In the figure 25, in the last line we are exporting the module to the app.js file. In other words we can say that we are including the modules into our app.js file. This app.js file will be the core part of our project.
Now when we run the program we get the following output:

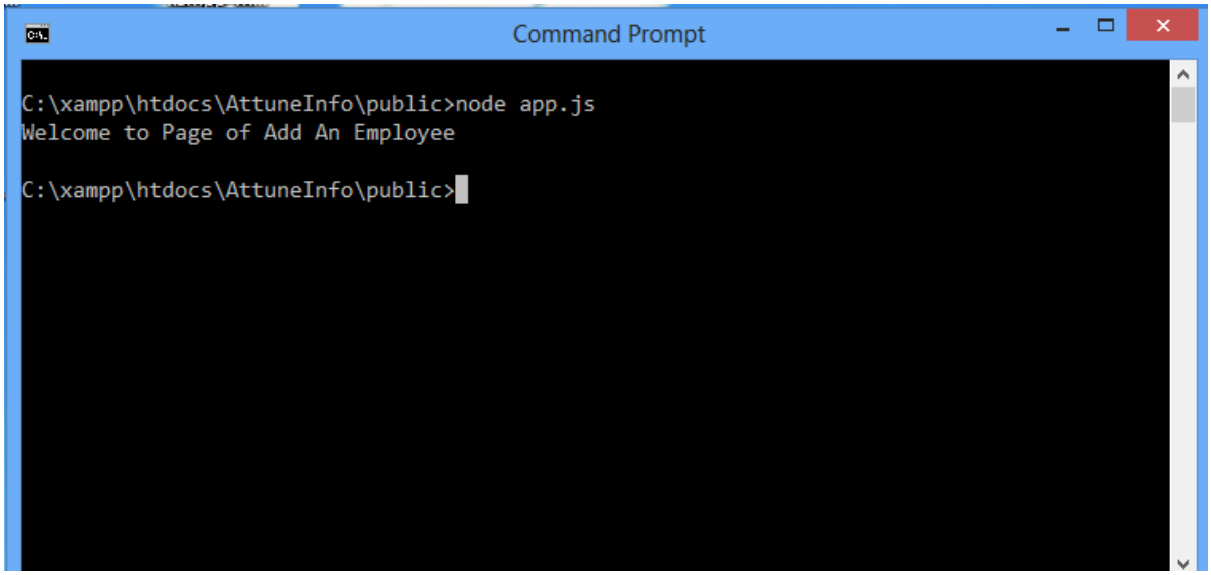

Figure 27 Output Employee Module

So this was the basic about the understanding of the modules. Now we will see some more examples about how to use modules in our project to make a great success.

9. More Modules

In our simple terminology, we are referring to the short cuts. There are different ways to make an application look and perform in different ways. In our scenario also we are making a small change in our “employee.js” file as shown in the below figure.

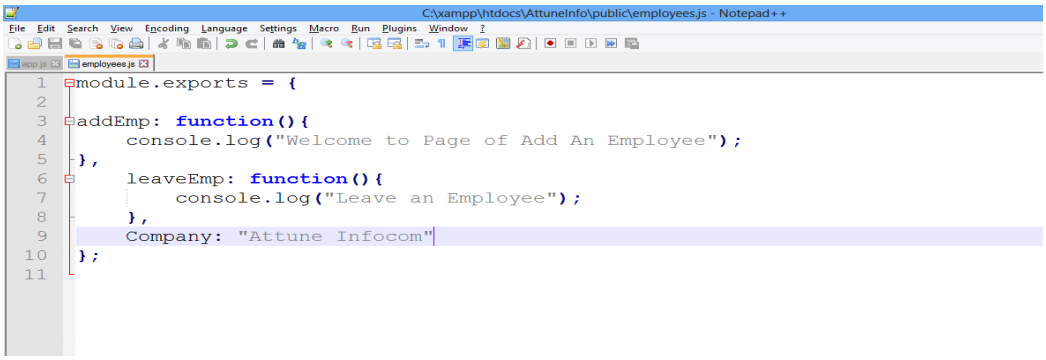

Figure 28 Modified Employee.js

In the previous section we have seen that we have included the module. exports in the last line but the modification that we have included in our js file is that we have made it a function and inside it we have included our two functionalities.

We do not have to make a change in our app.js file. So the desired output for the program will be as follows:

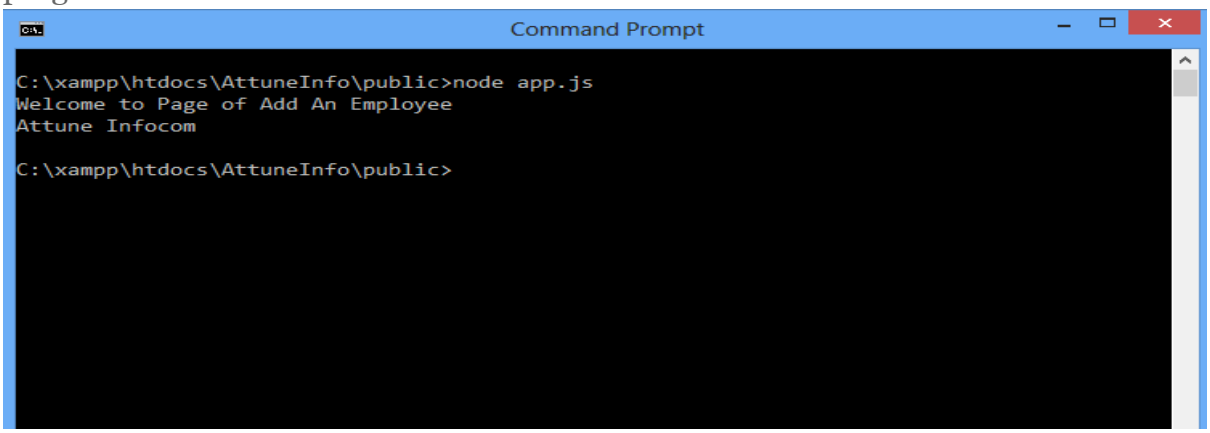

Figure 29 Output Modified Employee.js

10. Shared State Of Modules

In the previous sections we seen the basic and overall understanding of the modules. Now in this section, we will focus onto the activities in which one or more modules will share each other's state. In other words, they will use each other's functionalities.
Here we will make two different files namely "rajesh.js" and "suresh.js".

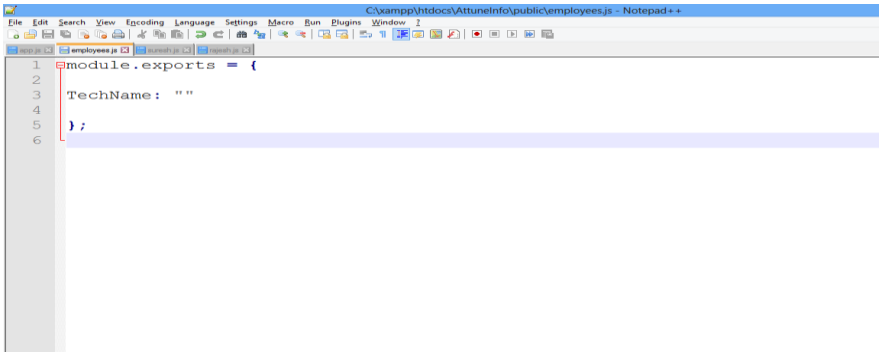

Figure 30 Shared States Employee.js

This diagram shows that we are currently working on only one parameter named "TechName". This "TechName" will hold the value of the technology that the employee is using.

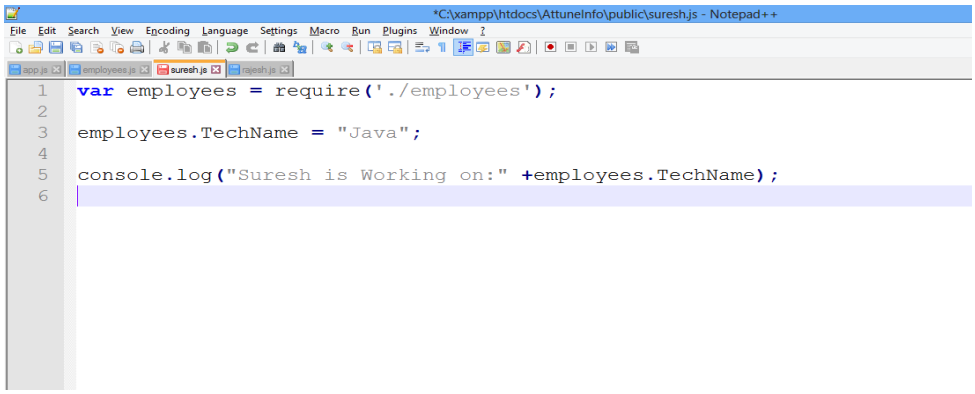

Figure 31 suresh.js

Figure 31 is handling the details of the employee named "suresh". His current technology that he is working on is "Java". So in the last line we have passed the information that the particular user is working on particular technology.

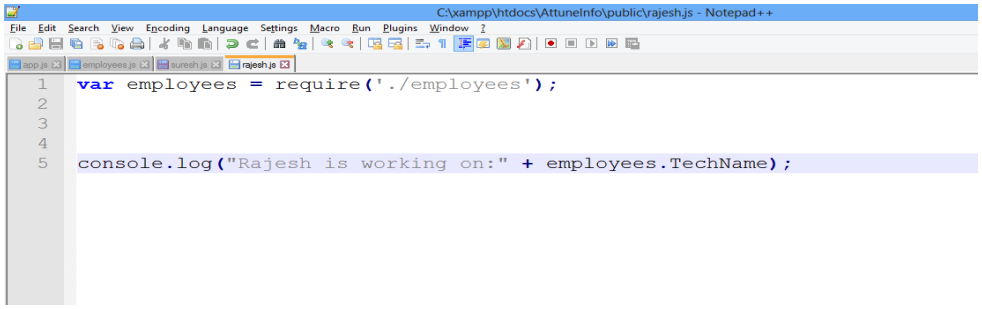

Figure 32 rajesh.js

Figure 31 is handling the details of the employee named “rajesh”. His current technology that he is working on not mentioned. So in the last line we have passed the information that the particular user is working on particular technology. So in this case the value that we have passed will be the same as that of the suresh as we can easily see in the output of it.

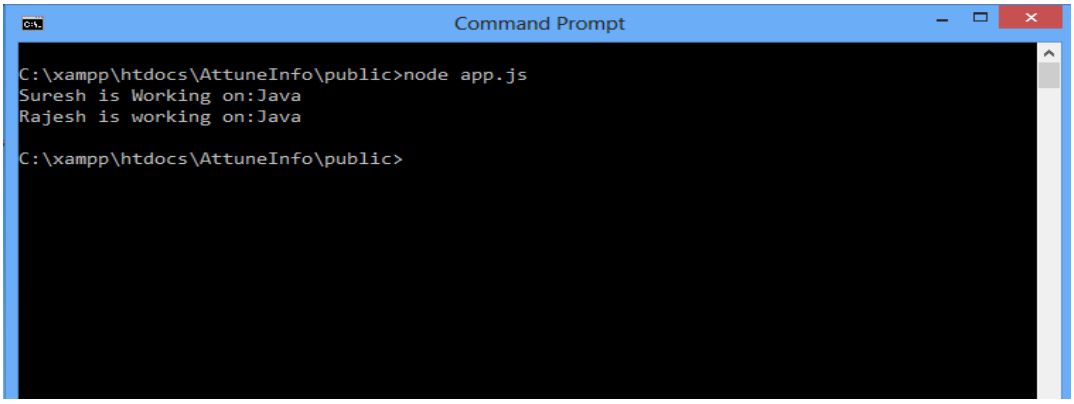

Figure 33 Output Shared State

11. Object Factory

Object factory is something like we are making a module a function and accordingly we are running that function. This is the same function that we have seen in the previous section. It is responding with the name of the technology of the employee.

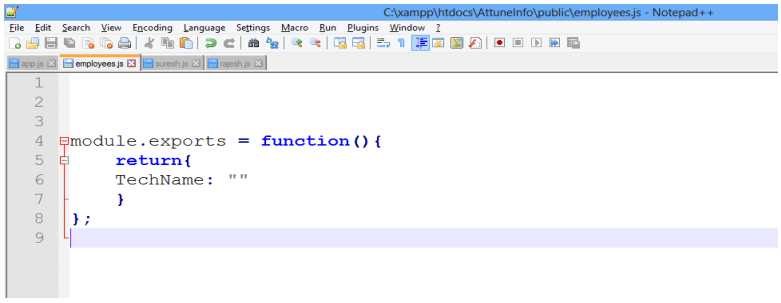

Figure 34 Object Factory (OF)

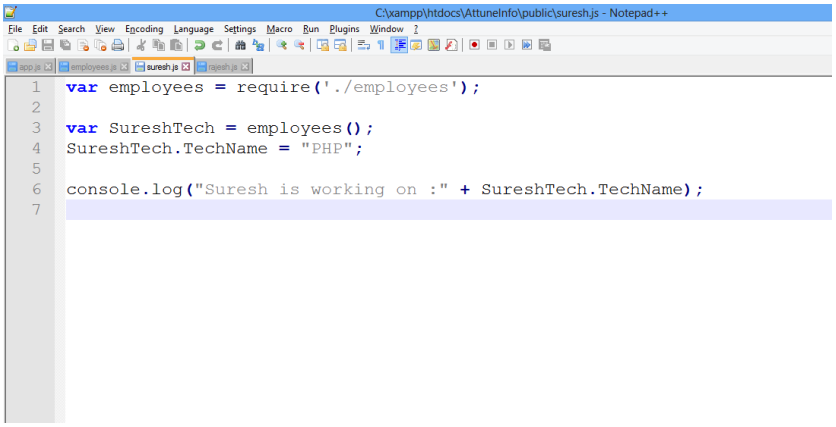

Figure 35 OF suresh.js

Here in this example we can see that suresh is working on the “PHP” technology.

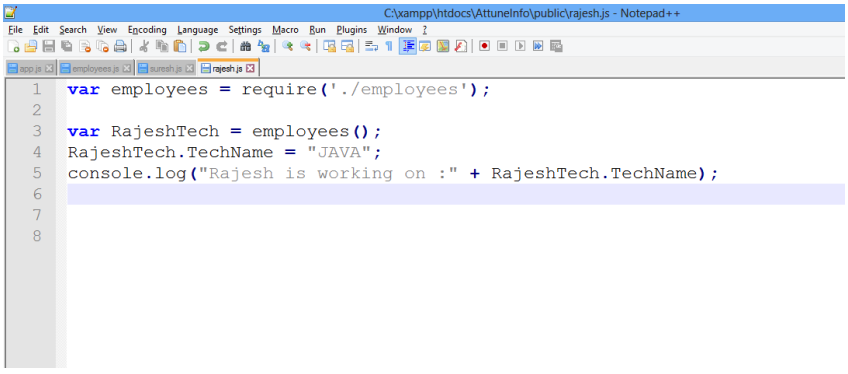

Figure 36 OF rajesh.js

Whereas in this example we can see that rajesh is working on “JAVA” technology.
So when we run this particular program the output will be:

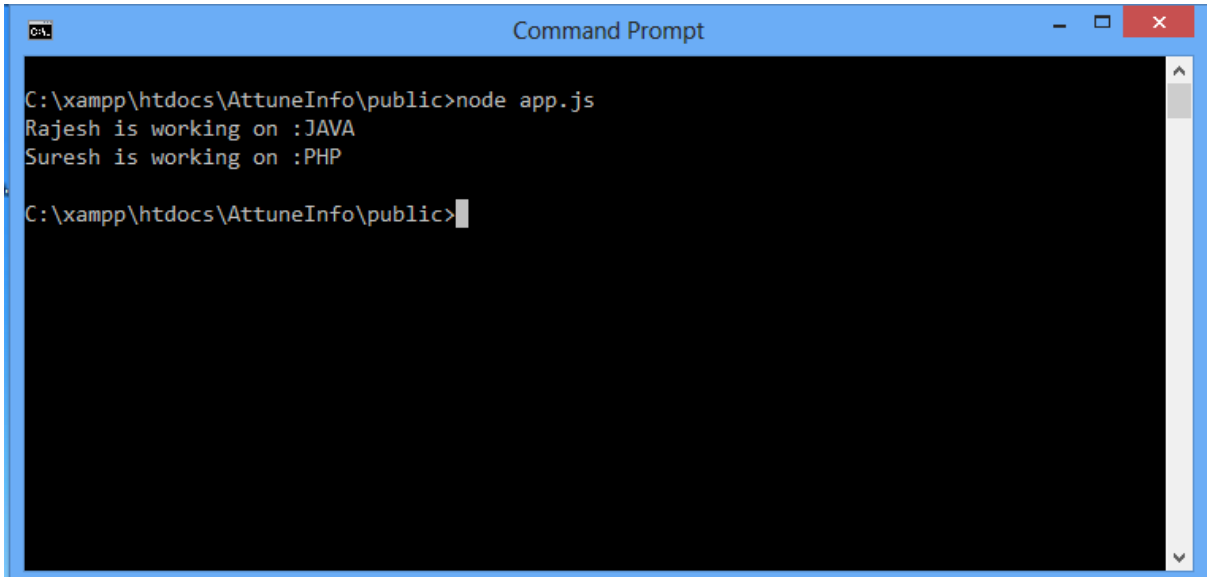

Figure 37 OF Output

Clearly we can see that both the employees are working on their respective technology.

12. Core Modules

Core module is the concept which is in-built module. So we just have to include it into our function without installing it from outside part. There are various core modules but in this section we will discuss the two main core modules namely (1) FS and (2) PATH.

12.1 fs

Fs is a module which is mainly responsible for handling the file-related activities. We can easily create a text file using javascript file and accordingly this file will be automatically saved into out desired folder.

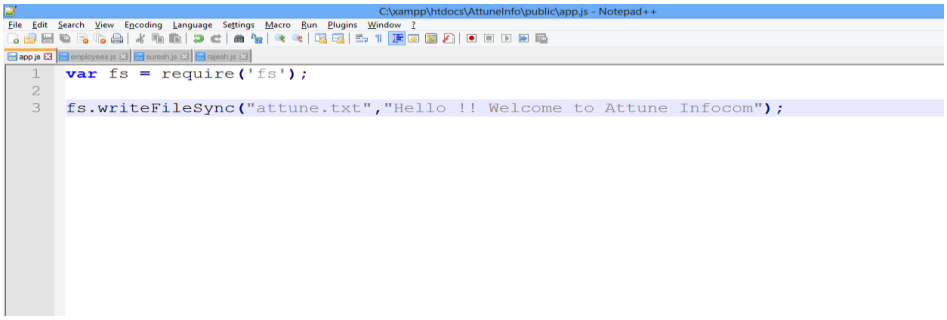

Figure 38 app.js

In the above diagram we can see that we have made a variable named “fs” and with the help of “fs” we are making a text file named “attune.txt”. This “attune.txt” file is having a value that is being included after a file name in “fs.writeFileSync()” function. So the desired output will be as follows:

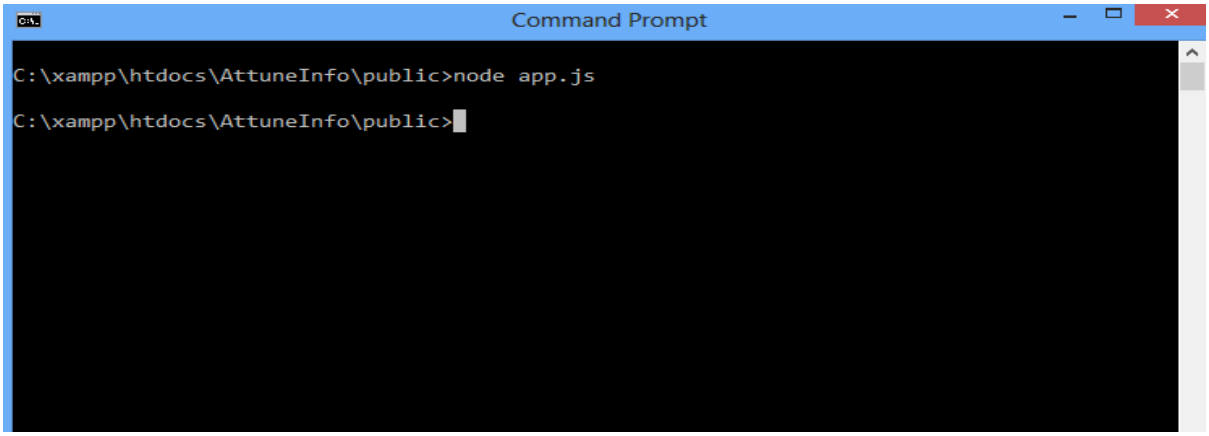

Figure 39 Running fs Program

When you run the program you will see that it will be moved to the specific folder as shown in the above diagram.

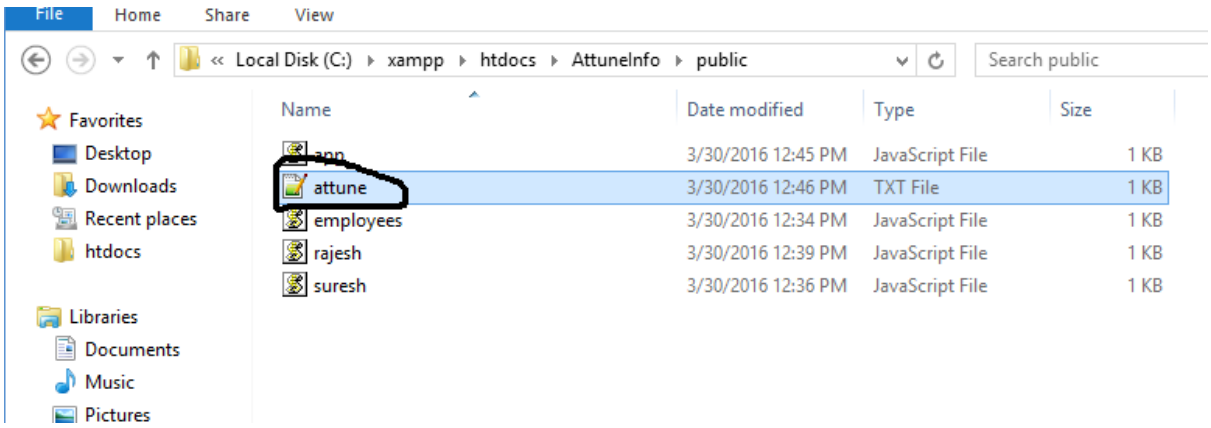

Figure 40 Folder Structure

Clearly we can see that "attune.txt" file has been made into the folder.

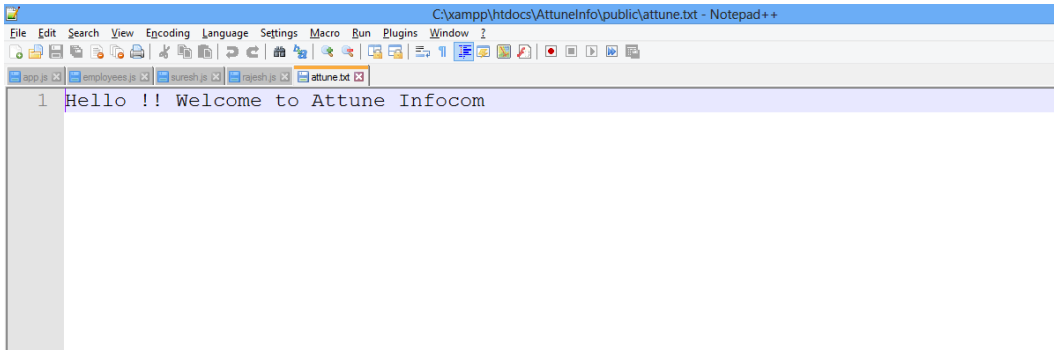

Figure 41 attune.txt

So when we open the "attune.txt" file we will see the content that we have passed into our fs.write() parameter as mentioned in the figure 38.

Now suppose we want to read the information that is contained inside our text file then we can also do with the help of "fs.readFileSync( )" as shown below:

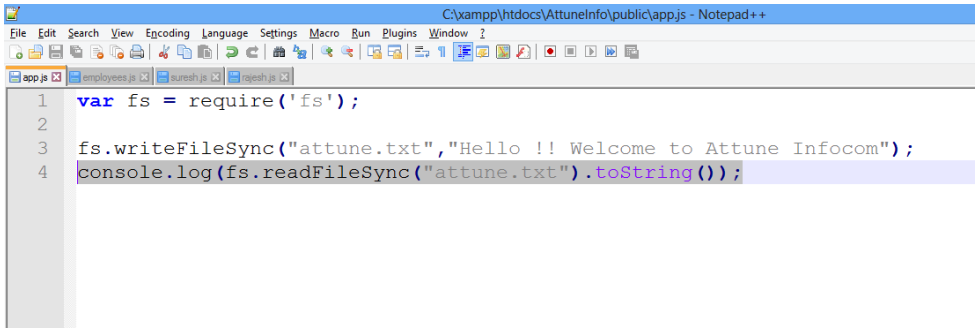

Figure 42 fs-Read a file

In the above diagram we can see that we are referring to “attune.txt” file and passing in the last “.tostring( )” which will return the information that has been contained inside the “attune.txt”.

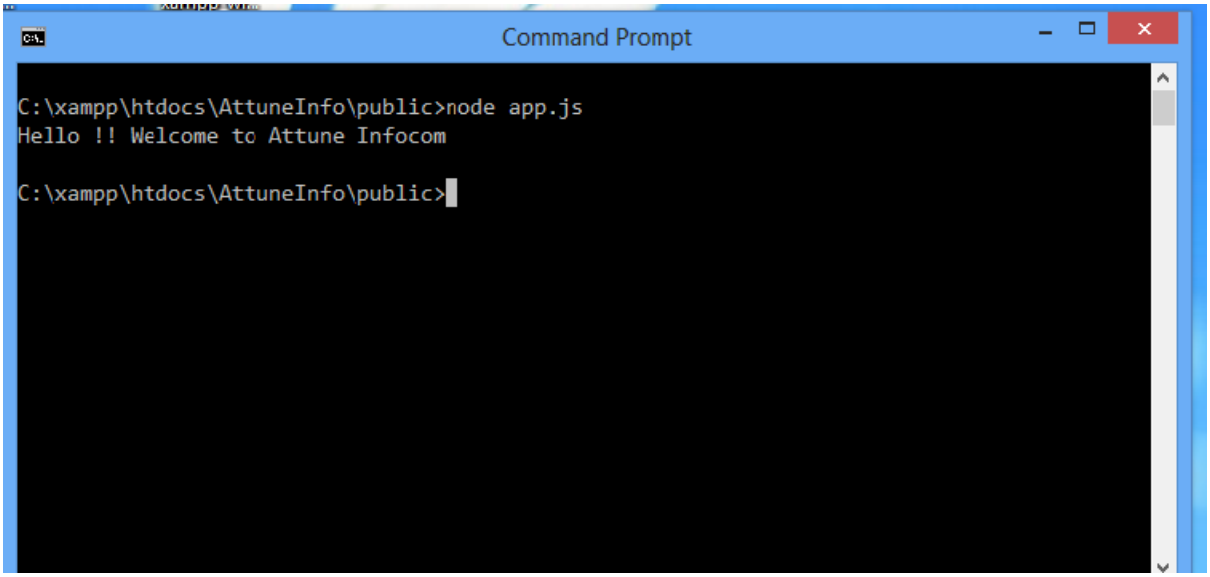

Figure 43 fs-Read Output

12.2 Path

Path is nothing but the location where our file is situated, so that we can also find the path. Let us see how?. With the help of the path module we can easily check the location of the file, directory name where our file is located and so on.

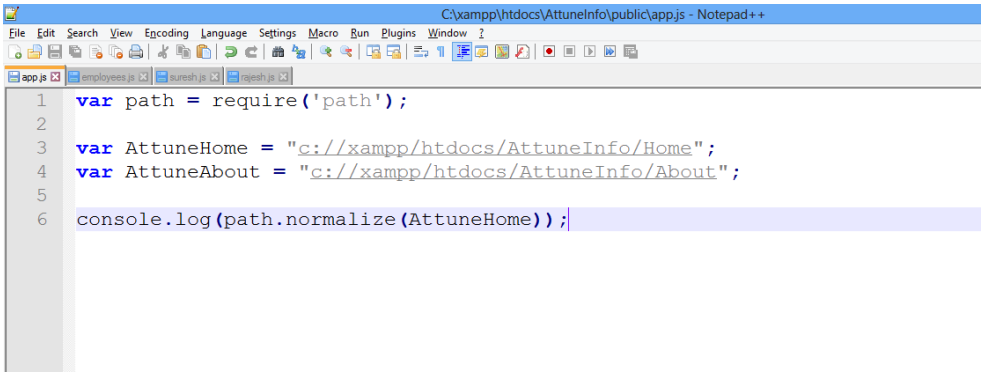

Figure 44 path

In this section, we are making two variables and passing globally values into it. After passing the values we are passing “path.normalize ( )” to see the basic path of the file.

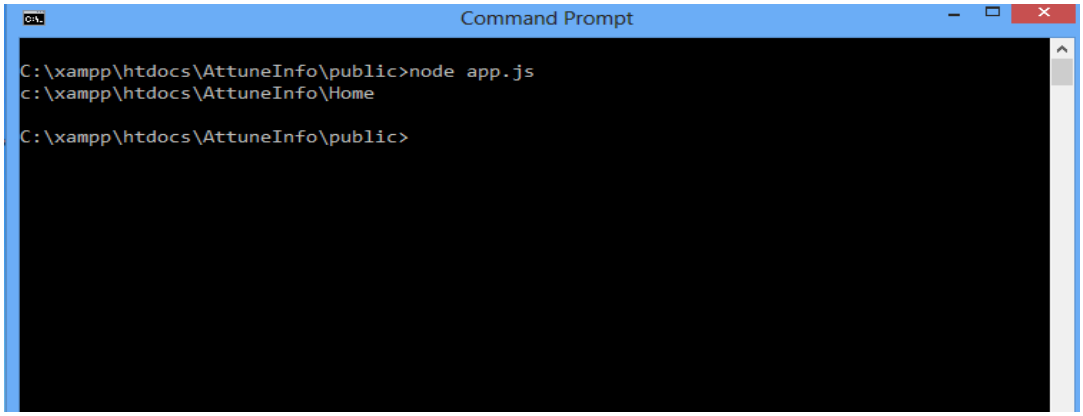

Figure 45 Output path

12.2.1 Some More Examples of Path

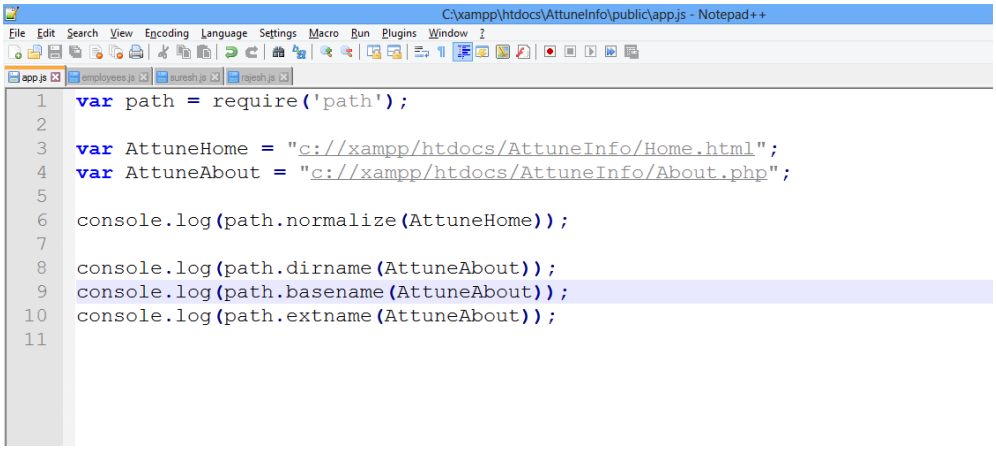

Figure 46 Example of Path

In the above example we can see that we have included some more functionalities regarding path. “dirname” will be responsible for giving the path of file where our file is located, “basename” will return the filename along with the extension and “extname” will return the extension of the file as shown in the output in figure below.

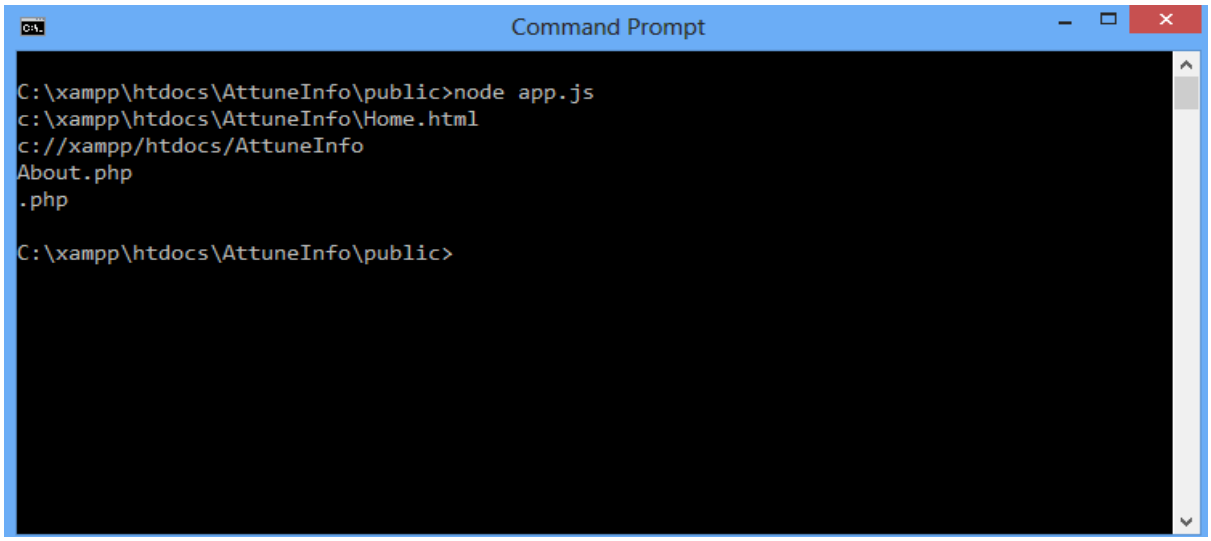

Figure 47 Output of path

13. Creating a basic server

As we know that server is nothing but the service provider to computer programs in the same computer or other computer. So in our node.js application, we are going to make a server and run the server to get the effective response from the output.

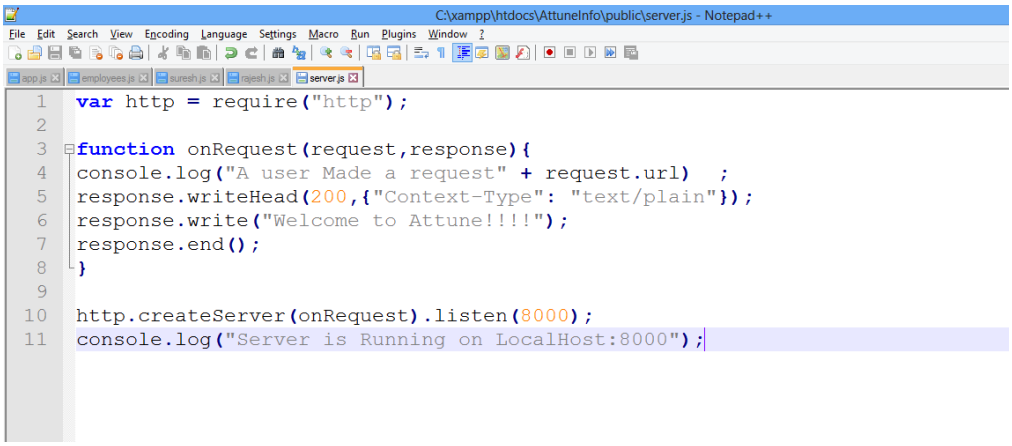

Figure 48 Example of basic server

Let us see hot this server works. As we know that server is always compatible with http, so we are including http in our server.js file. Simultaneously we are making a function to request some of the functions with two parameter one for requesting and one for responding.. First line of the function will display the url of the file that we are requesting and this information will be displayed in our command prompt whenever we will reflect or change the url. In this scenario we are only sending and requesting the text, so we are including the content-type as text/plain. “200” is the success message of the requested file. Finally we are running a text which will be displayed on to our browser.

The last two lines of the server.js file is the creating a server on the port number “8000” and displaying the text message that will be on the command prompt whenever we will run a success program.

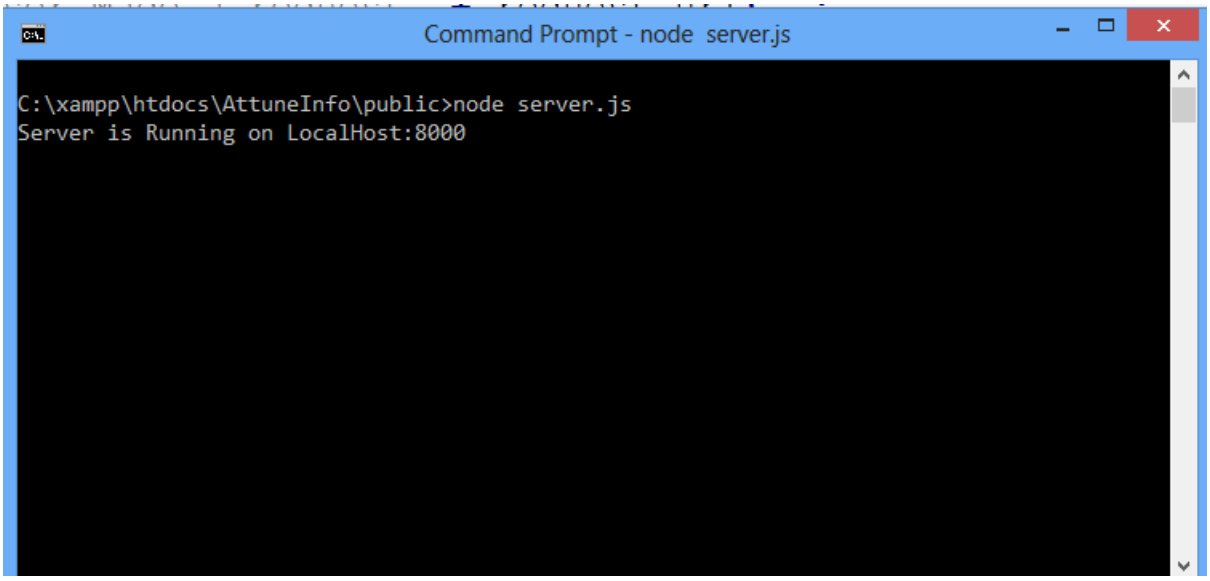

Figure 49 Running a basic server

This is running a basic server. Whenever we will run our server.js file and if our program is without any specific error then the command prompt will respond with a success message as shown in the above diagram.

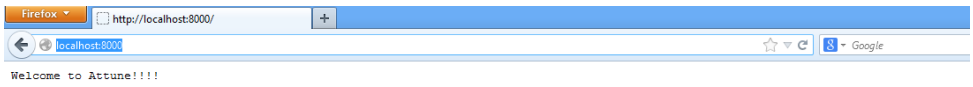

Figure 50 : Output of basic server

So when we open the browser and give the url as http://localhost:8000 our output will be displayed as shown in the above diagram.

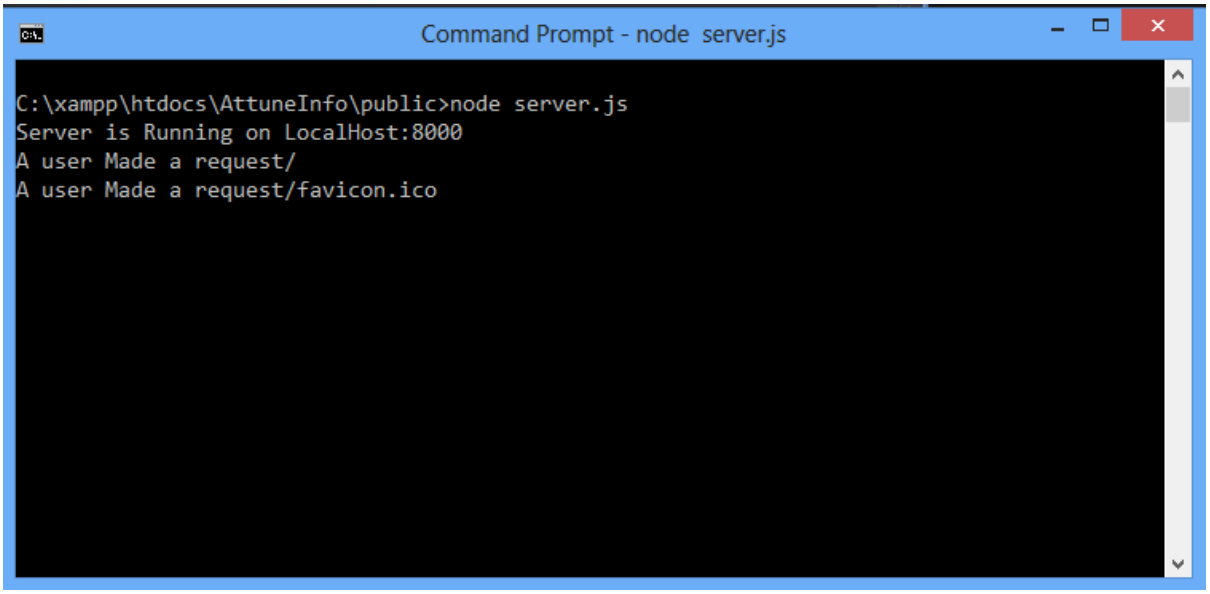

Figure 51 Output with Response

As we have declared in our server.js file that whenever we make a request to any of the url then that message will be displayed in to our command prompt. So when we run a first page then we will see a message that " A user made a request /" ."/" is nothing but the default page that we are seeing in our browser.

14. Simple Web File Server

In the previous section we have seen a basic server which was responding only a text. Now we will make a web page and with the help of server.js we will make a change into our web page file.

So first of all we will make one simple html file and will include that html file into our server file.

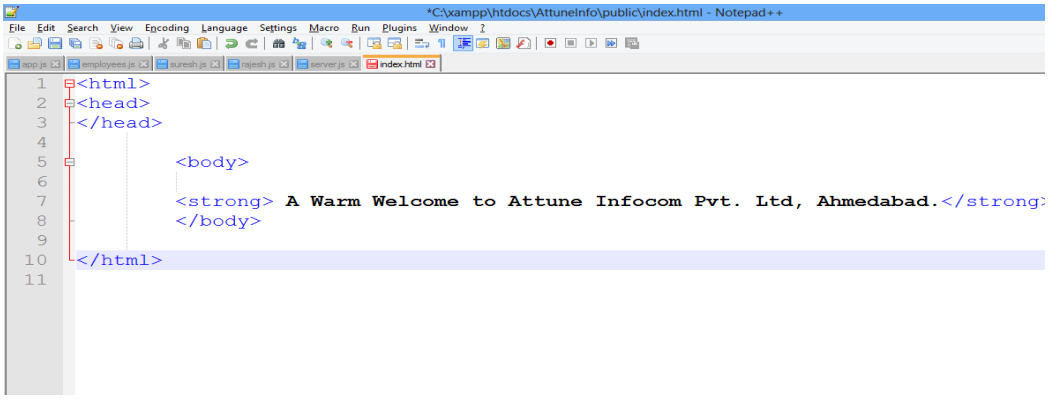

Figure 52 index.html

So here we have made a simple html file which will be displaying a message as depicted in html file. Remember we are just demonstrating the basic html file.

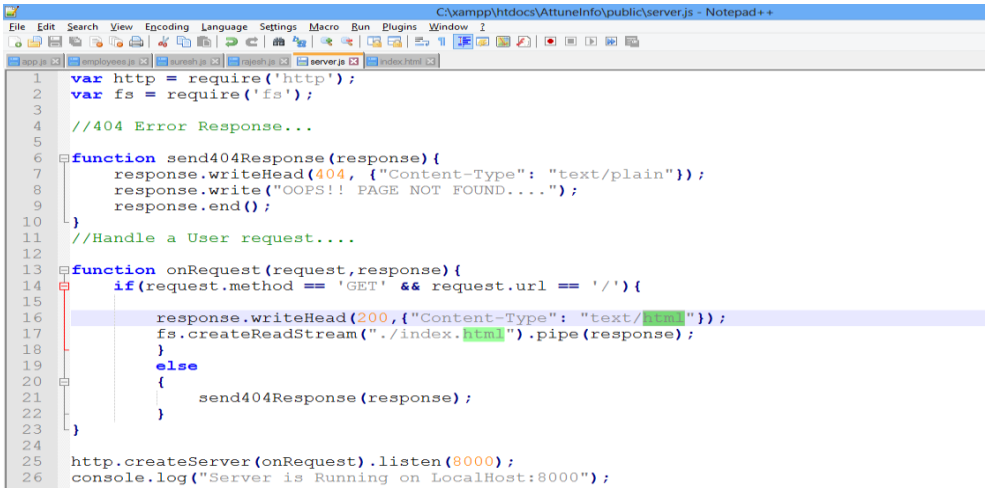

Figure 53 server.js

So we have here included a core module which is “fs” because we are now working with the files. We have included one function named “send404Response” which is nothing but the error that will be displayed whenever url does not find a page that has been requested.

Now we are making another request through a function to get a call to our html file. First of all we are checking with the requested method with “GET” and the url that we

want to display. So the default page that will be displayed in depicted in the last line of the function with the help of “fs.createReadStream()” . so in this function we are calling our “index.html” file.
The rest of the thing remains the same as we have discussed in the previous section.

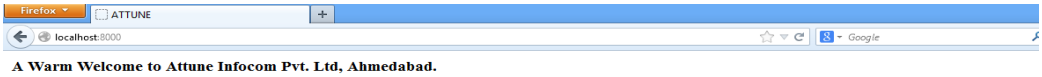

Figure 54 Output of web file server

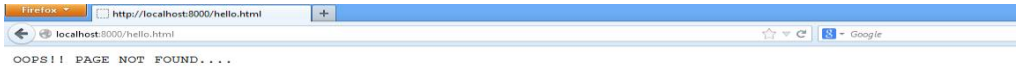

Figure 55 404 Error response Page

15. Connect

Connect is nothing but the module package that can be downloaded using our Node package Manager as discussed in the previous sections. With the help of connect module we are going to call different functions and run our specific server.

Let us see how to install connect module into our server. First of all move to the folder where our project is running. Give the specific command “npm install connect” to install connect module into our server.

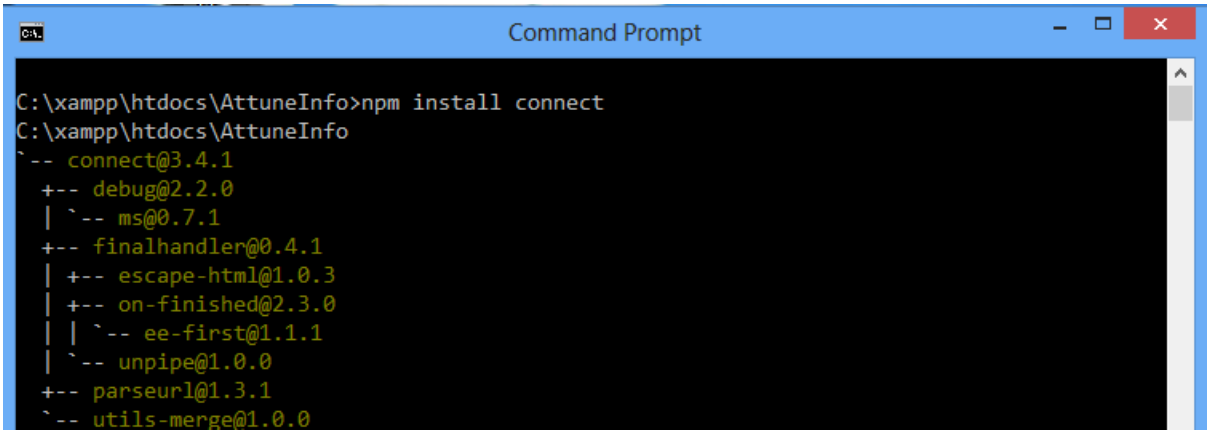

Figure 56 npm install connect

As we have successfully installed connect module, now we are going to use the connect module as a function and we will run the specific function of our project with the help of connect. Whenever we will install any of the packages “node_module” folder will be made into the project location and inside that “connect” folder will be available to verify that connect has been installed successful.

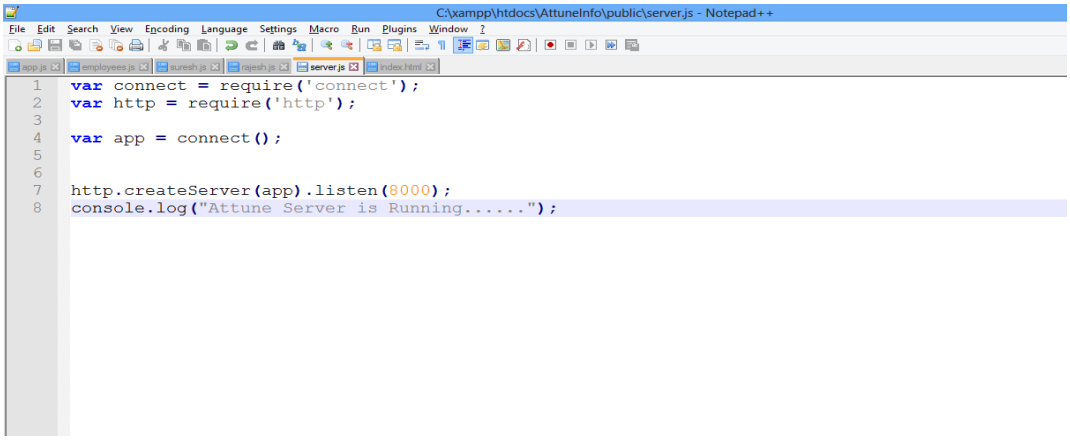

Figure 57 connect with no functions

As from the above figure we can see that we have included connect as a variable and we have used that variable to call it as a function with “app” variable. Now when we will run the “server.js” file the output will be:

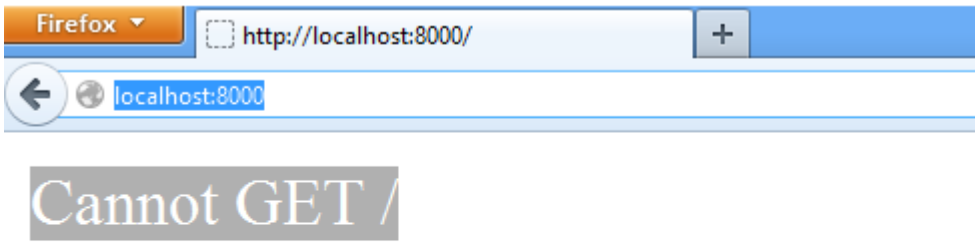

Figure 58 Connect with no Function

Now let us include some functions into our connect module. We will make two functions including “taskOne” and “taskTwo”. This two functions will import some message to display specific tasks.

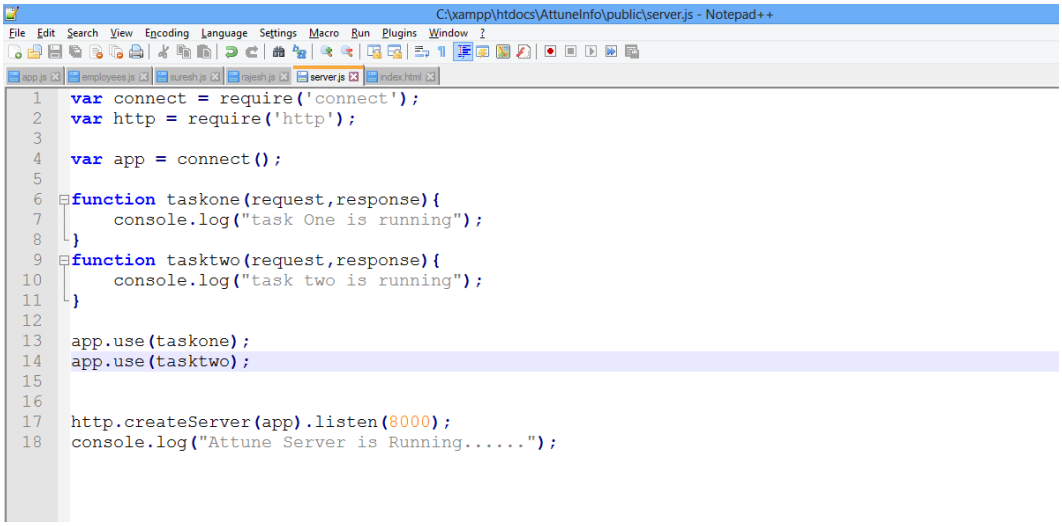

Figure 59 Connect with functions

As we can clearly see that we have included two simple functions, and we are using that functions with the help of “app.use( )” with the name of the functions as a parameter.

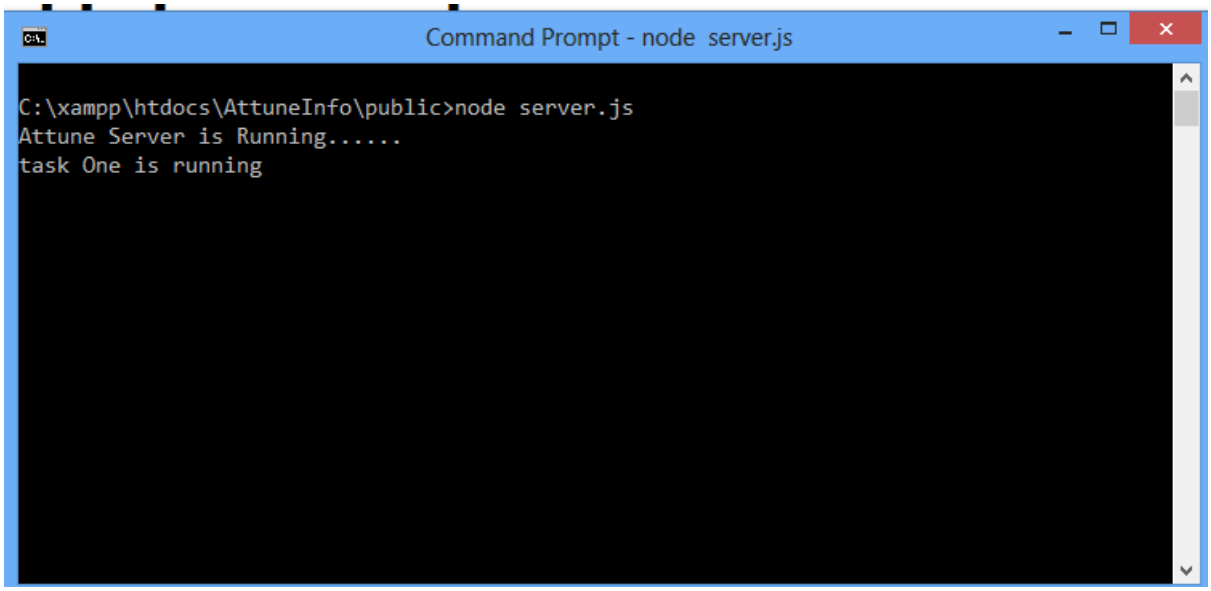

Figure 60 Output with functions

Now we are including one more parameter named next( ) which will execute the next function after running the current function.

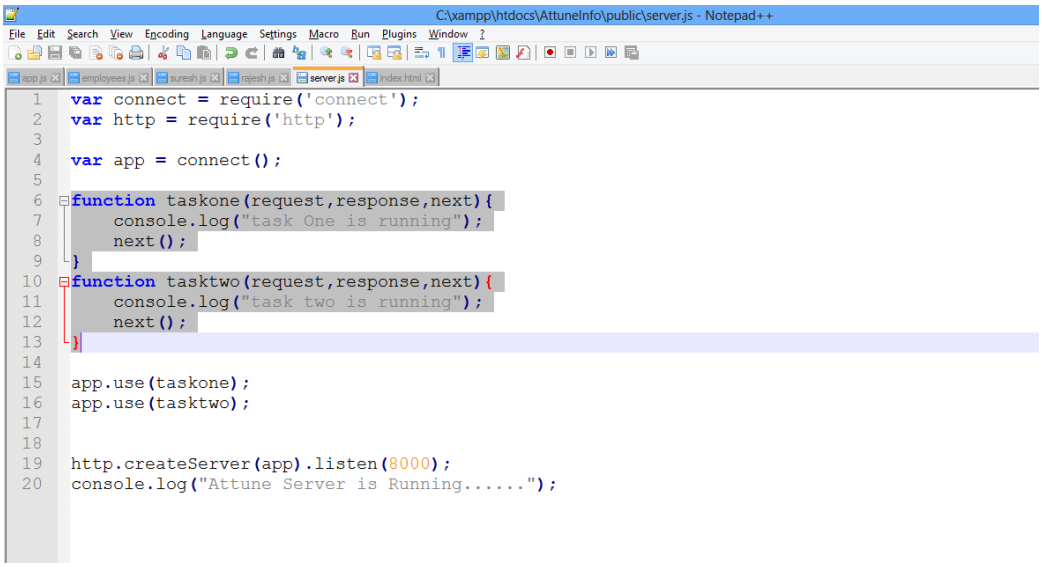

Figure 61 Connect with next( )

As we can see that we have included next() in both the functions, so whenever we will run a server we will get a desired output as:

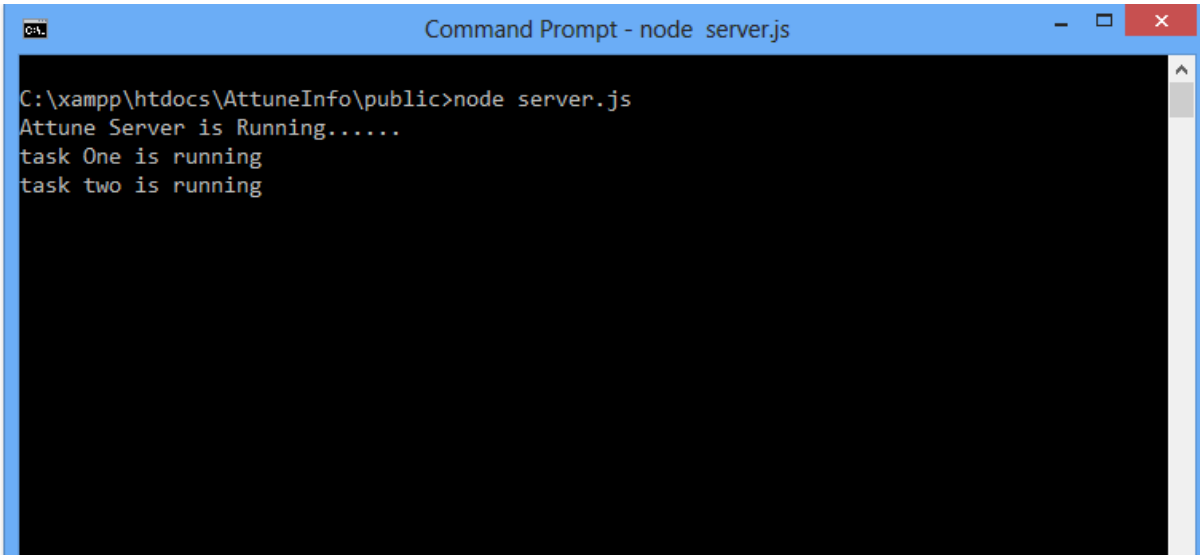

Figure 62 Output with next ()

16. Summary

So we have seen how to install the NodeJs framework and subsequently we have seen the different examples of Nodejs. We can easily make different web pages based on the server files. This guide was the intensive measure to understand the basics of NodeJs.