



Table of Contents

Introduction.....

What is Node.js

Installing Node.js.....

Node Core

Node Modules.....

The file system.....

The HTTP Modules.....

Node Package Manager

Web Server.....

Testing and Debugging.....

Automation and deployment.....

Introduction to Node.js:-

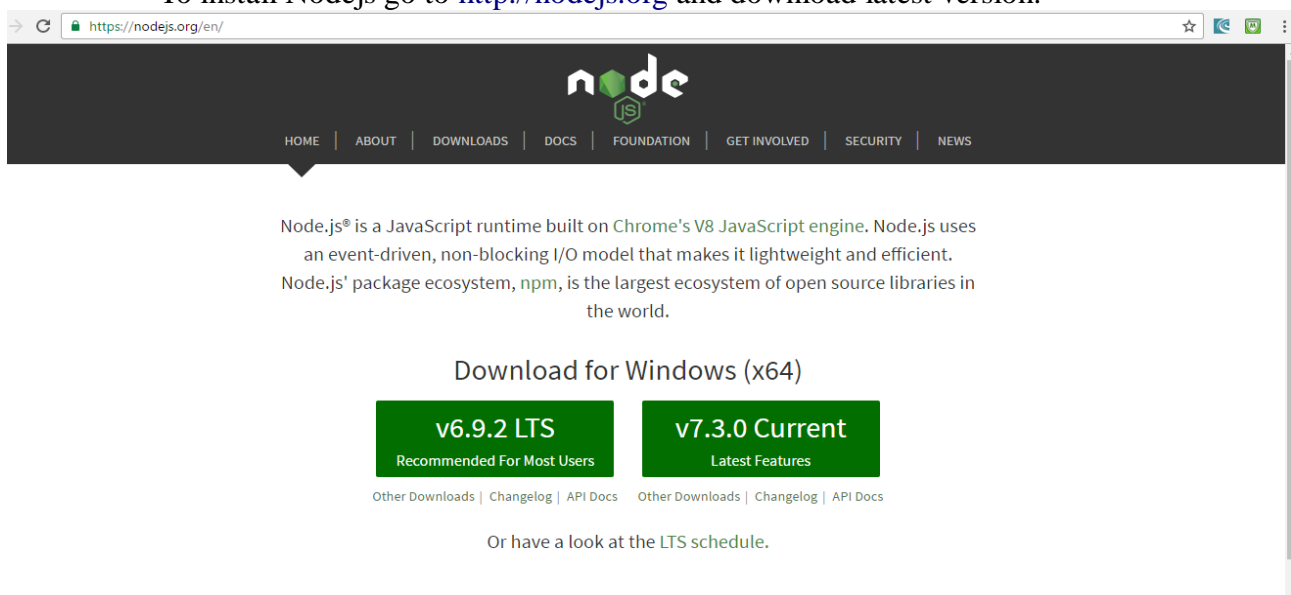
Node.js:- Nodejs allows us to run JavaScript application outside the scope of the browser, which means we can use JavaScript locally on our system or on large cluster on the cloud. Nodejs is single threaded, means all users use single thread so there is no blocking. Nodejs is asynchronous, means it can do more than one thing at a time.

What you should know:-

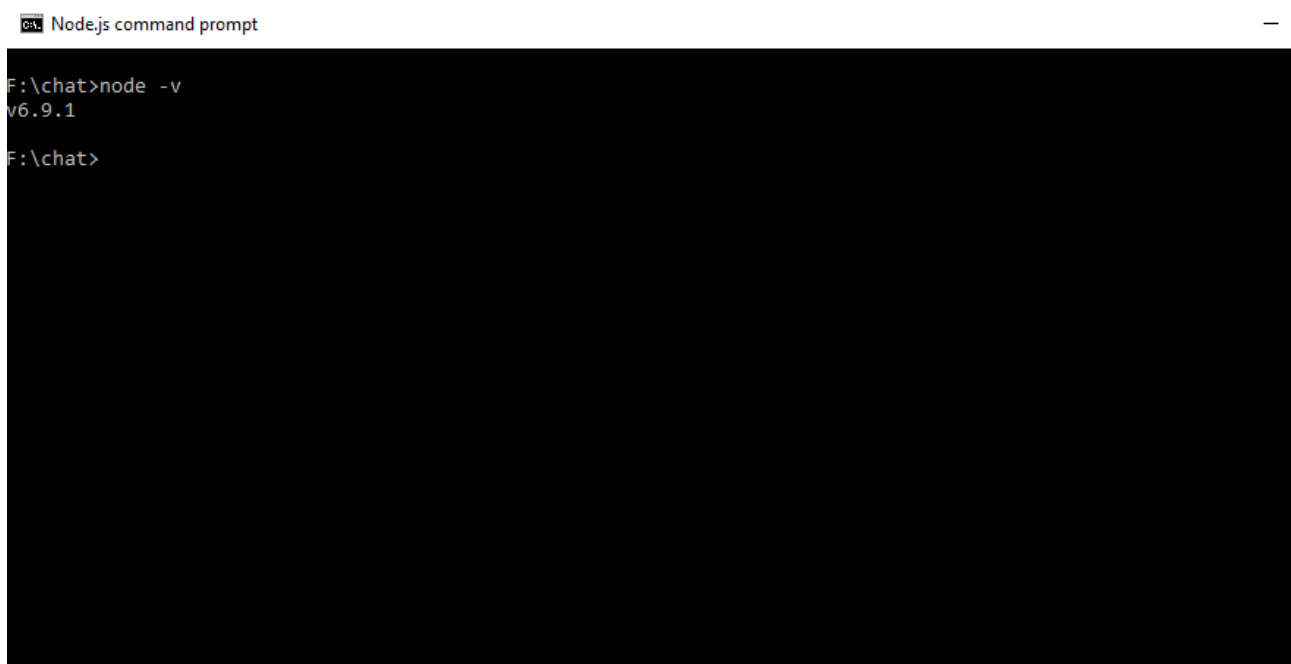
1. JavaScript
2. command line

Installing Node.js:-

To install Nodejs go to <http://nodejs.org> and download latest version.



To check version of Nodejs go to command prompt and type “node -v”.

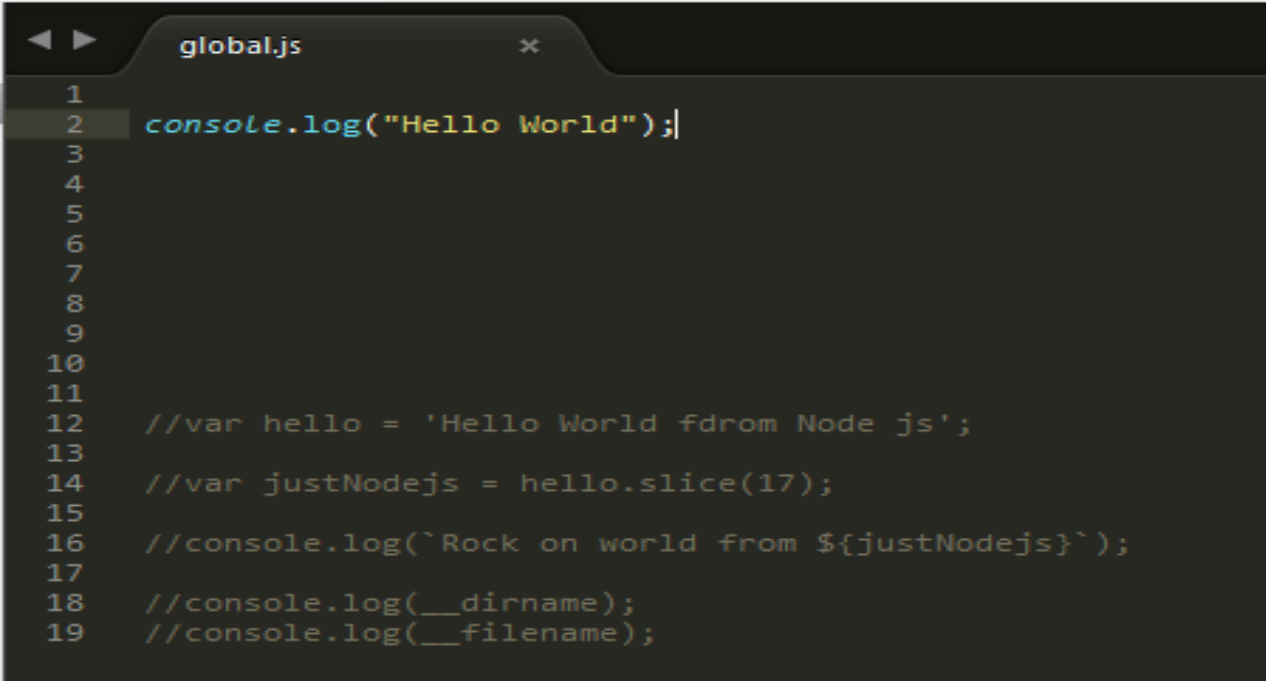


Node Core:-

The Global object:- Global objects mean we can use them without any requirement in Nodejs global, process, console etc. for more check Nodejs documentation.

Example1:- Create a folder named start and inside the folder create a js file named global.js then type

`console.log("Hello World")` or `global.console.log("Hello World")`

A screenshot of a code editor window with a dark theme. The window has a single tab titled 'global.js' with a close button 'x' on the right. The code is written in JavaScript and is as follows:

```
1
2 console.log("Hello World");|
3
4
5
6
7
8
9
10
11
12 //var hello = 'Hello World fdrom Node js';
13
14 //var justNodejs = hello.slice(17);
15
16 //console.log(`Rock on world from ${justNodejs}`);
17
18 //console.log(__dirname);
19 //console.log(__filename);
```

Then go to nodejs terminal window and select your folder and type

`node global.js` or `node global`

```
Node.js command prompt
Your environment has been set up for using Node.js 6.9.1 (x64) and npm.
C:\Users\Tek2>e:
E:\>cd "Exercise Files"
E:\Exercise Files>cd Ch03
E:\Exercise Files\Ch03>cd 03_01
E:\Exercise Files\Ch03\03_01>cd start
E:\Exercise Files\Ch03\03_01\start>node global.js
Hello World
E:\Exercise Files\Ch03\03_01\start>
```

And you will see **Hello World** on terminal.

Example2:- Inside global.js file type

```
var hello = "Hello World from Nodejs";
```

```
console.log(hello);
```

```
global.js
1 var hello = 'Hello World from Node js';
2
3 console.log(hello);
4
```

Output on terminal:- **Hello World from Nodejs.**

```
Node.js command prompt
Your environment has been set up for using Node.js 6.9.1 (x64) and npm.
C:\Users\Tek2>e:
E:\>cd "Exercise Files"
E:\Exercise Files>cd Ch03
E:\Exercise Files\Ch03>cd 03_01
E:\Exercise Files\Ch03\03_01>cd start
E:\Exercise Files\Ch03\03_01\start>node global.js
Hello World
E:\Exercise Files\Ch03\03_01\start>node global.js
Hello World fdrom Node js
E:\Exercise Files\Ch03\03_01\start>
```

Example3:- Inside global.js type

```
var hello = "Hello World from Nodejs",
var justNode = "hello.slice(17)";
```

To show justNode on terminal output we have to use template string (``) as follow

```
console.log(`Rock on world from ${justNode}`);
```

```
global.js
1  var hello = 'Hello World fdrom Node js';
2
3  var justNodejs = hello.slice(17);
4
5  console.log(`Rock on world from ${justNodejs}`);
6
7  //console.log(__dirname);
8  //console.log(__filename);
```

Then go to terminal and type

“node global”

Output on terminal:-

Rock on world from Nodejs

```
Node.js command prompt

E:\Exercise Files\Ch03\03_01\start>node global.js
Rock on world from Node.js

E:\Exercise Files\Ch03\03_01\start>
```

Example4:- Inside global.js file

```
console.log(__dirname);
console.log(__filename);
```

`__dirname` gives full path of directive where file located in and `__filename` gives full path of directive where file located in with the file name.

```
global.js

1 var hello = 'Hello World from Node.js';
2
3 var justNodejs = hello.slice(17);
4
5 console.log(`Rock on world from ${justNodejs}`);
6
7 console.log(__dirname);
8 console.log(__filename);
```

Then in terminal type:- `node global.js`

```
Node.js command prompt

E:\Exercise Files\Ch03\03_01\start>node global.js
Rock on world from Node.js
E:\Exercise Files\Ch03\03_01\start
E:\Exercise Files\Ch03\03_01\start\global.js
E:\Exercise Files\Ch03\03_01\start>
```

require :-

Require function is globally available, require function is used to import other Nodejs module.

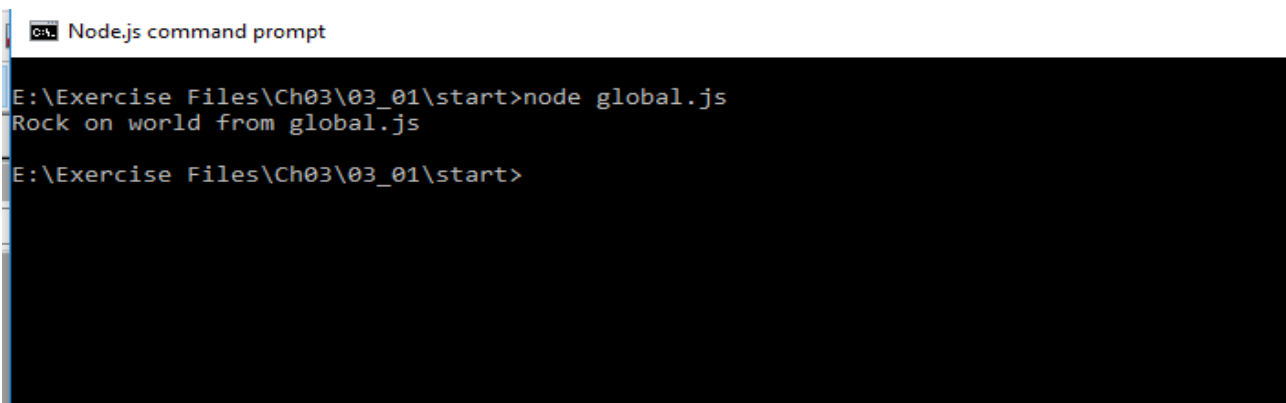
```
var path = require("path");
```

The above code import path module in our file so we can use some tools of path module to work with path. Using "path.basename(__filename)" we unplug the file name from full path to file.

A screenshot of a code editor window titled 'global.js'. The code is as follows:

```
1 var path = require("path");
2
3
4 console.log(`Rock on world from ${path.basename(__filename)}`);
5
```

On typing" node global" in terminal

A screenshot of a Node.js command prompt window. The title bar says 'Node.js command prompt'. The command prompt shows the following sequence of text:

```
E:\Exercise Files\Ch03\03_01\start>node global.js
Rock on world from global.js
E:\Exercise Files\Ch03\03_01\start>
```

Process object:-

1. The process object is globally available so we can use it anywhere in our application. Process object contains functionalities which allow us to interact with current process instance.

2. One of the things which we can do with process object is that we can collect all the information from terminal window when the application starts. All of that information will be saved in a variable called process.argv which stands for argument variable used to start the process.

Example:- Create a folder named start and include a file app.js then type

```
console.log(process.argv);
```


A screenshot of a code editor window titled 'app.js'. The editor shows a single line of code: `console.log(process.argv);` on line 1. The line numbers 1 through 15 are visible on the left margin. The code is written in a light blue font on a dark background.

Process is a global object so we can use it where we like. Then go to terminal window and type “node app”.

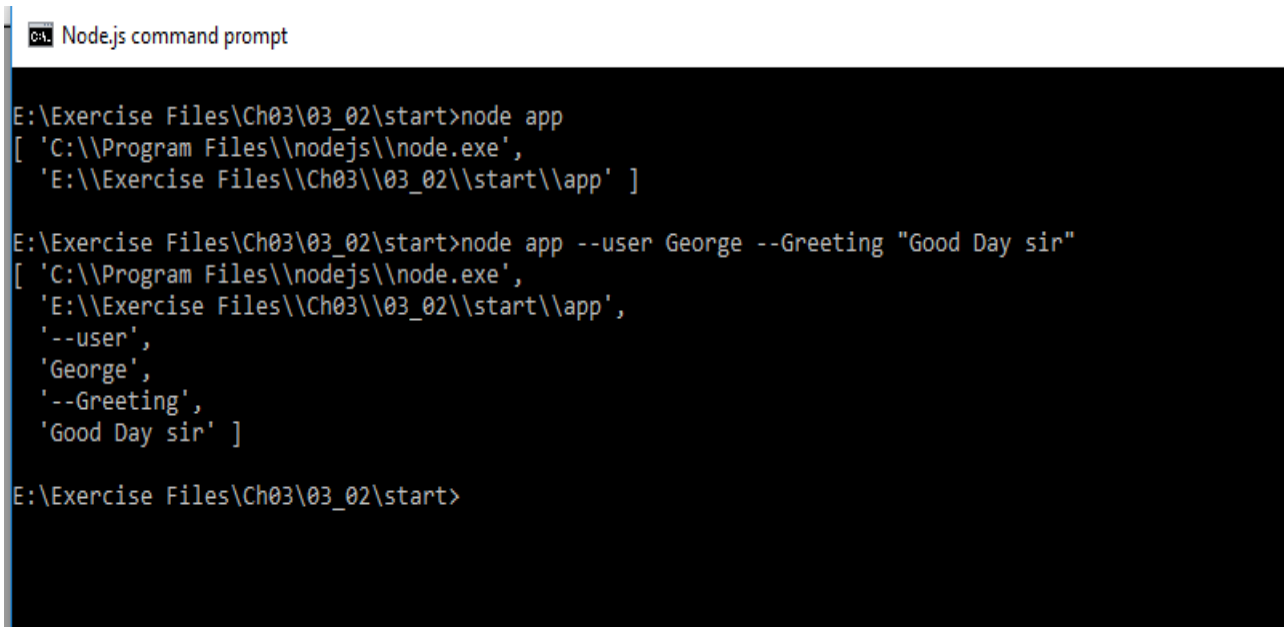
A screenshot of a Node.js command prompt window. The title bar says 'Node.js command prompt'. The command prompt shows the following sequence of text:
E:\Exercise Files\Ch03\03_02\start>node app
['C:\\Program Files\\nodejs\\node.exe',
 'E:\\Exercise Files\\Ch03\\03_02\\start\\app']
E:\Exercise Files\Ch03\03_02\start>

As we see process.argv is an Array which contains path to node application and path to app.js file.

We can add more information in process.argv array by adding additional information in terminal window as follow

node app –user George –Greeting “Good Day Sir”

It gives an array containing these additional information as shown below.



```
Node.js command prompt

E:\Exercise Files\Ch03\03_02\start>node app
[ 'C:\\Program Files\\nodejs\\node.exe',
  'E:\\Exercise Files\\Ch03\\03_02\\start\\app' ]

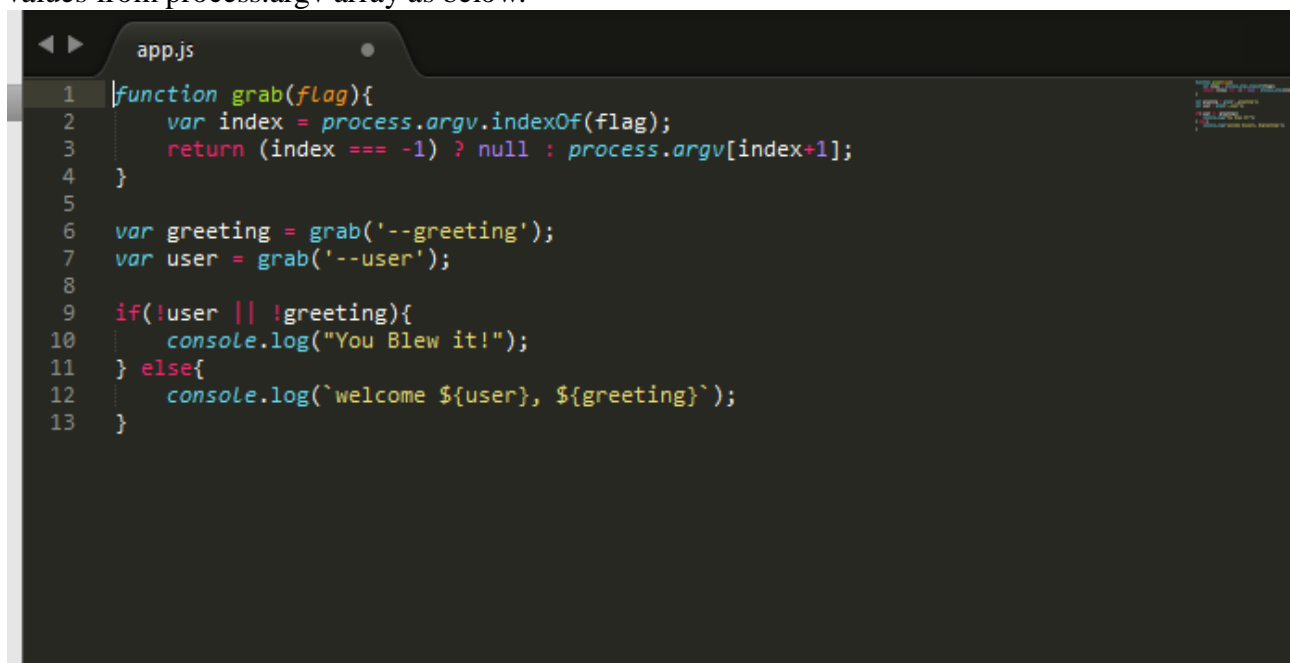
E:\Exercise Files\Ch03\03_02\start>node app --user George --Greeting "Good Day sir"
[ 'C:\\Program Files\\nodejs\\node.exe',
  'E:\\Exercise Files\\Ch03\\03_02\\start\\app',
  '--user',
  'George',
  '--Greeting',
  'Good Day sir' ]

E:\Exercise Files\Ch03\03_02\start>
```

As `process.argv` is an array so we can use it as JavaScript array and can perform array methods on it.

Example:-

Now go to text editor create `app.js` file in start folde and write a grab function to retrieve values from `process.argv` array as below.



```
app.js
1 function grab(flag){
2     var index = process.argv.indexOf(flag);
3     return (index === -1) ? null : process.argv[index+1];
4 }
5
6 var greeting = grab('--greeting');
7 var user = grab('--user');
8
9 if(!user || !greeting){
10     console.log("You Blew it!");
11 } else{
12     console.log(`welcome ${user}, ${greeting}`);
13 }
```

Then go to terminal window open your start folder and type “node app”.

```
Node.js command prompt

E:\Exercise Files\Ch03\03_02\start>node app
You Blew it!

E:\Exercise Files\Ch03\03_02\start>
```

As we didn't add user and greeting information so output is **You Blew it!**. So add user and greeting information as shown below and run it.

```
Node.js command prompt

E:\Exercise Files\Ch03\03_02\start>node app --user Alex --greeting "Hello Hello Hello"
welcome Alex, Hello Hello Hello

E:\Exercise Files\Ch03\03_02\start>
```

Standard input and Standard output:-

Standard input and standard output are features of process object. These two objects offer us a way to communicate with process while it is running. Now we are using these object to read and write data to the terminal.

Example:- Create a start folder and include a file named ask.js then open that file in text editor and write code as below.

```
process.stdout.write("Hello");
process.stdout.write("World\n\n\n") ;
```

```
ask.js
1 process.stdout.write("Hello");
2 process.stdout.write("World \n\n\n");
3
4
5
6
7
8
9 |
10
11
12
13
14
15
16
17
18
```

Then go to terminal window and run by typing “node ask”

```
Node.js command prompt
E:\Exercise Files\Ch03\03_03\start>node ask
HelloWorld

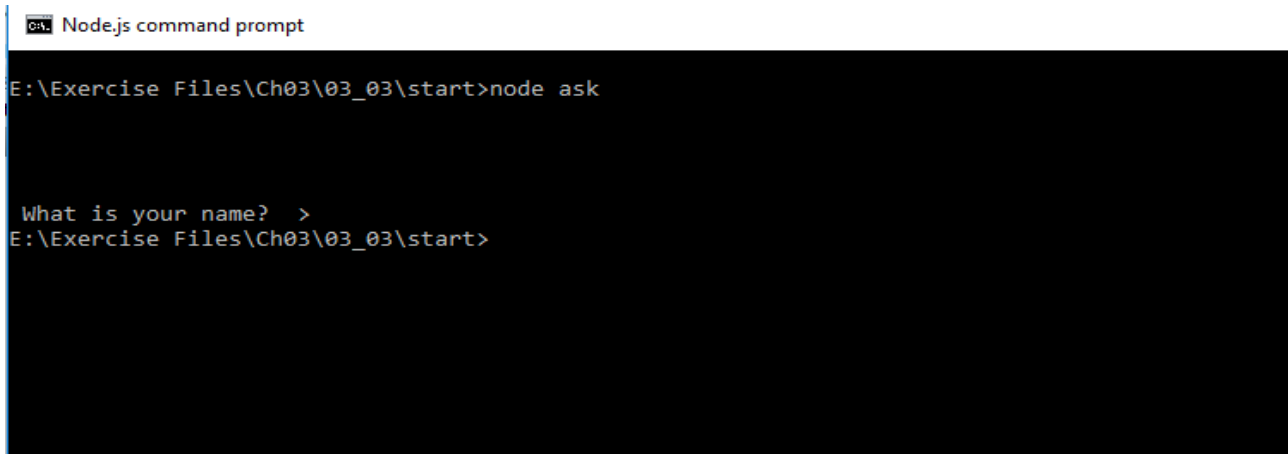
E:\Exercise Files\Ch03\03_03\start>
```

Example:-

In this example we are going to create an array containing some short of questions and use a function ask() to ask the question. So open the ask.js in text editor and create an array name question and a function ask as shown below.

```
ask.js
1 var questions = [
2     "What is your name?",
3     "What is your fav hobby?",
4     "What is your preferred programming language?"
5 ];
6
7 var answers = [];
8
9 function ask(i){
10     process.stdout.write(`\n\n\n\n ${questions[i]}`);
11     process.stdout.write(" > ");
12 }
13 |
14 ask(0);
15 /*process.stdin.on('data', function(data
```

Then go to terminal and run by typing “node ask” and output will be as follow



```
Node.js command prompt
E:\Exercise Files\Ch03\03_03\start>node ask

What is your name? >
E:\Exercise Files\Ch03\03_03\start>
```

As we see output, question is being asked but after asking the question terminal exit it does not put new line etc. so we need a listener to listen the answer to the question .

So go back to text editor and add a event listener as below

```
process.stdin.on('data', function(data){

    process.stdout.write('\n' + data.toString().trim() + '\n');

});
```



```
ask.js
1  var questions = [
2      "What is your name?",
3      "What is your fav hobby?",
4      "What is your preferred programming language?"
5  ];
6
7  var answers = [];
8
9  function ask(i){
10     process.stdout.write(`\n\n\n\n ${questions[i]}`);
11     process.stdout.write(" > ");
12 }
13
14
15 process.stdin.on('data', function(data){
16
17     process.stdout.write('\n' + data.toString().trim() + '\n');
18 });
19
20 ask(0);
21 /*process.on('exit', function(){
22     process.stdout.write('\n\n\n\n\n');
```

On running the app using “node ask”

```
Node.js command prompt - node ask

E:\Exercise Files\Ch03\03_03\start>node ask

What is your name? >
```

As we see terminal window does not exit, means it is expecting some input, in this example answer to the question. App is still running so we are using nodejs asynchronously here and it is waiting for answer, when we provide an answer it we handle a callback function asynchronously.

Now after giving input it will echo and show the answer on terminal and app will continuously listen so to stop the app we have to type Ctrl+C.

Use cls command to clear the command window.

```
Node.js command prompt

E:\Exercise Files\Ch03\03_03\start>node ask

What is your name? > Alex

Alex
hello hello

hello hello
^C
E:\Exercise Files\Ch03\03_03\start>
```

Now instead of echo the answer on terminal window we are going to save the answers in answer array and we check if there is any more question as follow.

```
Process.stdin.on('data', function(data){
```

```

        answer.push(data.toString().trim());

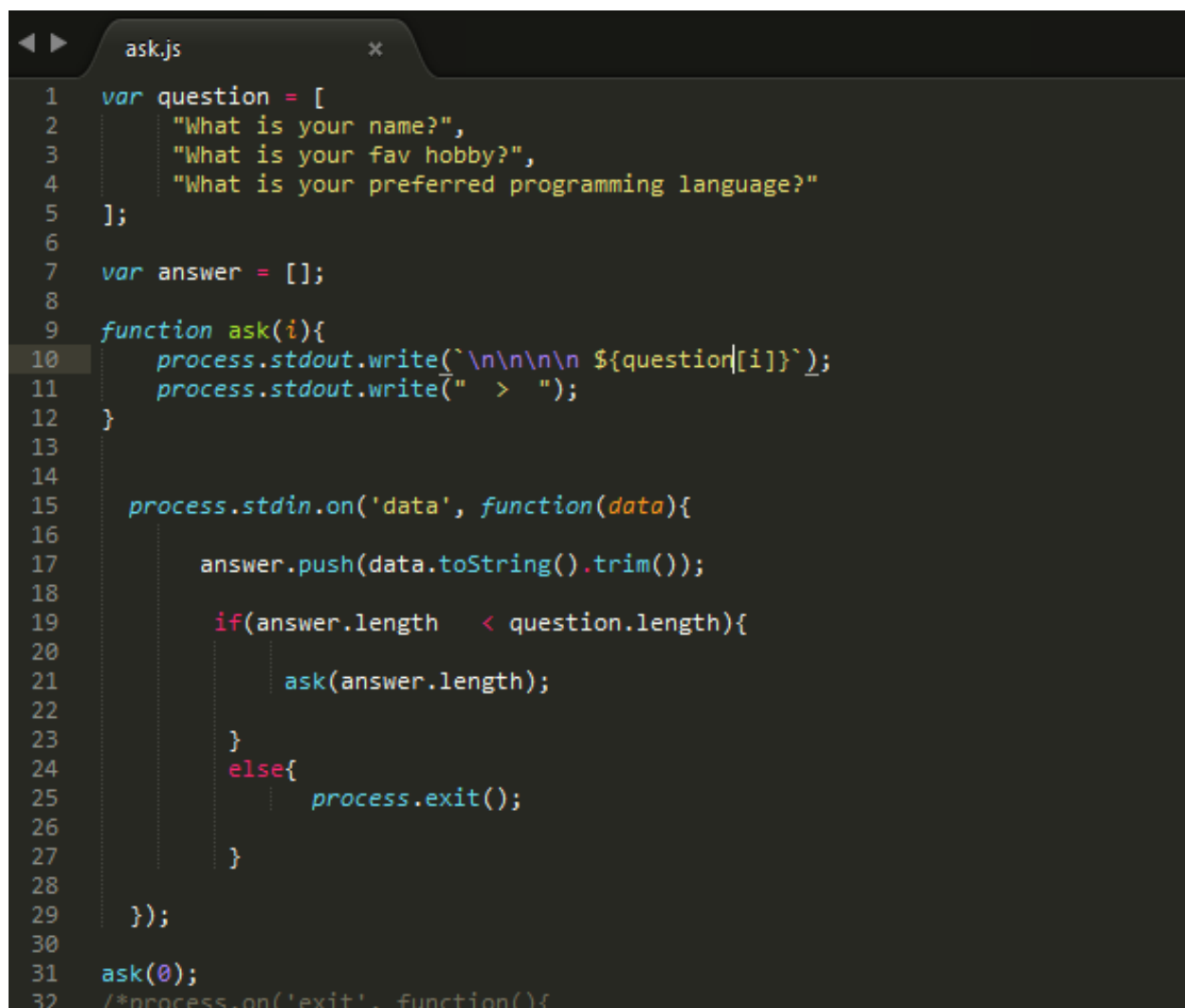
        if(answer.length < question.length){

            ask(answer.length);

        }
        else{
            process.exit();
        }

    });

```



```

1  var question = [
2      "What is your name?",
3      "What is your fav hobby?",
4      "What is your preferred programming language?"
5  ];
6
7  var answer = [];
8
9  function ask(i){
10     process.stdout.write(`\n\n\n\n ${question[i]}`);
11     process.stdout.write(" > ");
12 }
13
14
15 process.stdin.on('data', function(data){
16
17     answer.push(data.toString().trim());
18
19     if(answer.length < question.length){
20
21         ask(answer.length);
22
23     }
24     else{
25         process.exit();
26     }
27
28 });
29
30
31 ask(0);
32 /*process.on('exit', function(){

```

Then go to terminal and run the app by “node ask”

```
E:\Exercise Files\Ch03\03_03\start>node ask

What is your name? > Alex

What is your fav hobby? > cricket

What is your preferred programming language? > javascript
E:\Exercise Files\Ch03\03_03\start>
```

After third question answered `process.exit()` invoked and terminal exist.

Now instead of exist the terminal we want to show all the answers on screen so go to text editor and use another event listener as follow

```
ask.js
1 var question = [
2   "What is your name?",
3   "What is your fav hobby?",
4   "What is your preferred programming language?"
5 ];
6
7 var answer = [];
8
9 function ask(i){
10   process.stdout.write(`\n\n\n\n ${question[i]}`);
11   process.stdout.write(" > ");
12 }
13
14 process.stdin.on('data', function(data){
15   answer.push(data.toString().trim());
16   if(answer.length < question.length){
17     ask(answer.length);
18   }
19   else{
20     process.exit();
21   }
22 }
23
24 });
25
26 process.on('exit', function(){
27   process.stdout.write(`\n\n\n`);
28   process.stdout.write(`Go play! ${answer[1]} ${answer[0]} you can finish writing ${answer[2]} later`);
29   process.stdout.write(`\n\n\n`);
30 });
31
32 ask(0);
```

Then go to terminal window and run the app using “node ask” as follow.


```
E:\Exercise Files\Ch03\03_03\start>node ask

What is your name? > Alex

What is your fav hobby? > cricket

What is your preferred programming language? > javascript

Go play cricket Alex you can finish writing javascript later

E:\Exercise Files\Ch03\03_03\start>
```

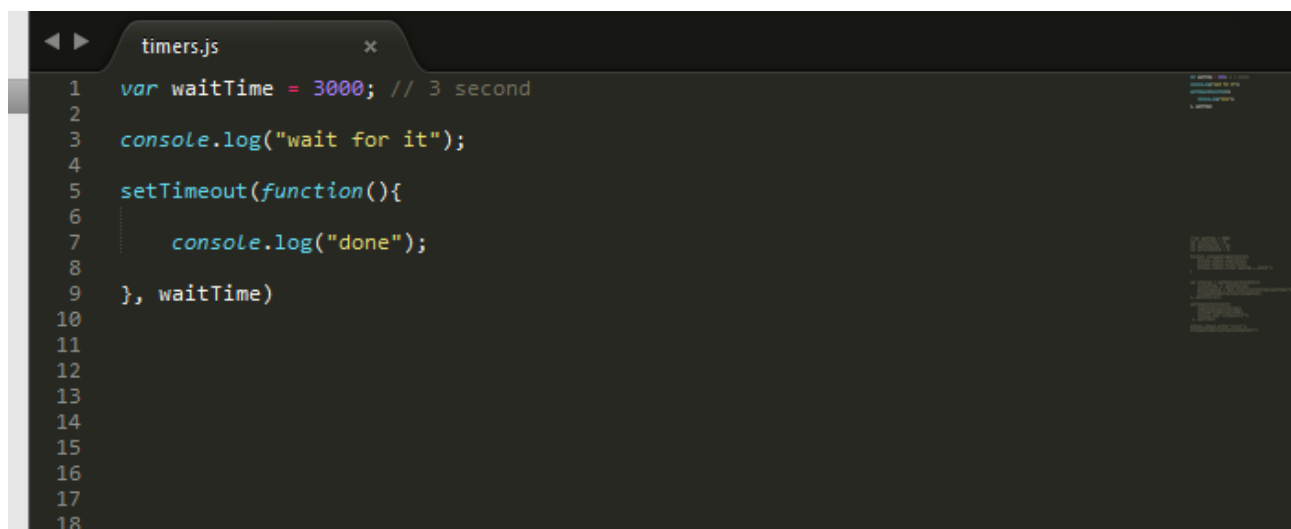
Note:- Using event listener we use Node.js asynchronously.

Global timing function:-

Timing function is globally available in Node.js and it is also used in Node.js asynchronously.

Example:- `setTimeout(cb, ms)`, `clearTimeout(t)` etc. for more check out Node.js documentation.

Example:- Create a folder named `start` and include a file named `timer.js` then open that file in text editor and write code for `setTimeout()` function as follows.

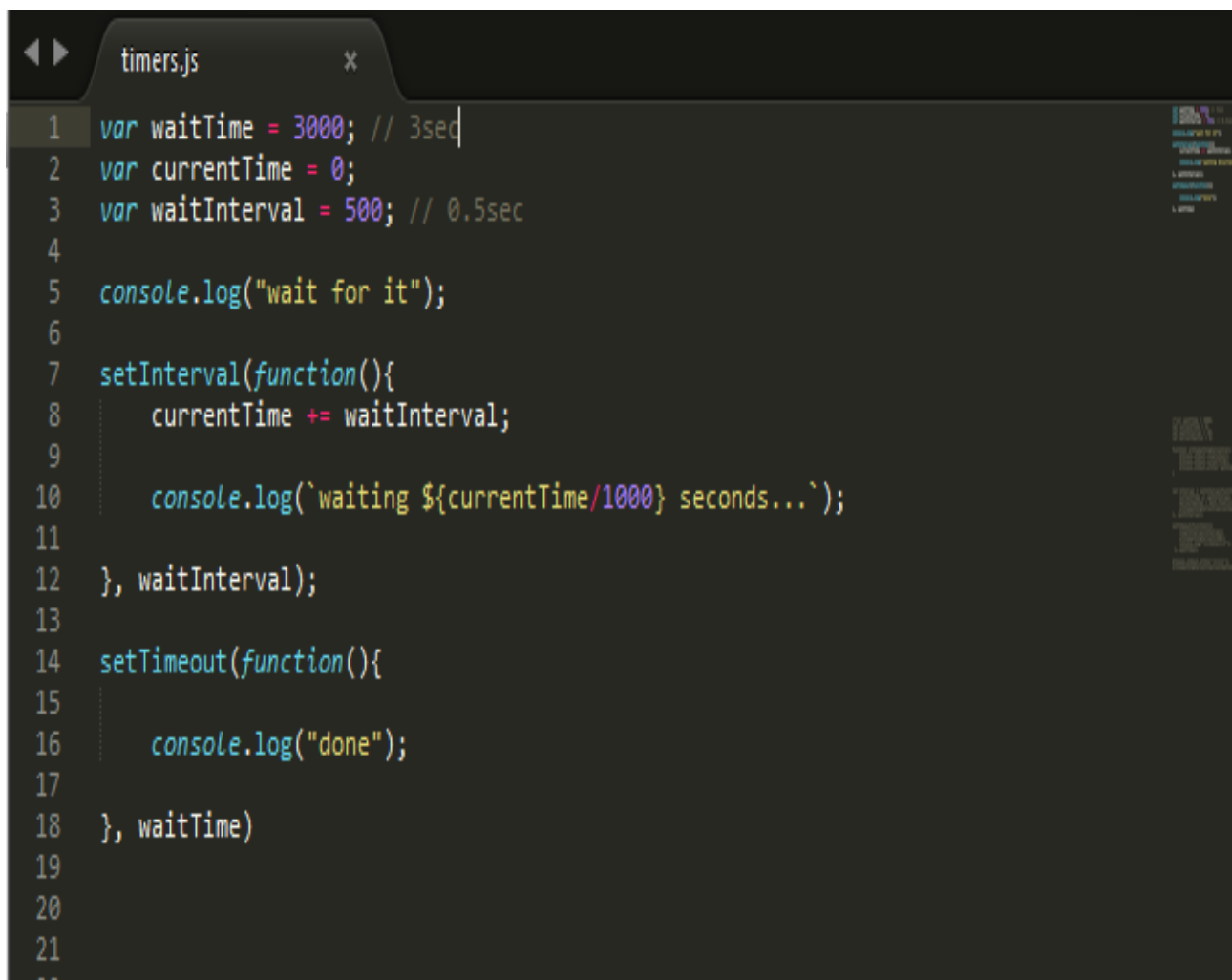


```
timers.js
1  var waitTime = 3000; // 3 second
2
3  console.log("wait for it");
4
5  setTimeout(function(){
6
7      console.log("done");
8
9  }, waitTime)
10
11
12
13
14
15
16
17
18
```

Here in the above code `waitTime` is 3000 milliseconds so `setTimeout` will wait for 3 sec and then respond back as shown below.

```
E:\Exercise Files\Ch03\03_04\start>node timers
wait for it
done
E:\Exercise Files\Ch03\03_04\start>
```

Next we will use `setInterval()` function to set the interval to fire that function as follow



```
timers.js
1 var waitTime = 3000; // 3sec
2 var currentTime = 0;
3 var waitInterval = 500; // 0.5sec
4
5 console.log("wait for it");
6
7 setInterval(function(){
8     currentTime += waitInterval;
9
10    console.log(`waiting ${currentTime/1000} seconds...`);
11
12 }, waitInterval);
13
14 setTimeout(function(){
15     console.log("done");
16
17 }, waitTime)
18
19
20
21
22
```

Then go to terminal and run the app using “node timers”

```
E:\Exercise Files\Ch03\03_04\start>node timers
wait for it
waiting 0.5 seconds...
waiting 1 seconds...
waiting 1.5 seconds...
waiting 2 seconds...
waiting 2.5 seconds...
done
waiting 3 seconds...
waiting 3.5 seconds...
waiting 4 seconds...
waiting 4.5 seconds...
waiting 5 seconds...
waiting 5.5 seconds...
waiting 6 seconds...
waiting 6.5 seconds...
waiting 7 seconds...
waiting 7.5 seconds...
waiting 8 seconds...
waiting 8.5 seconds...
waiting 9 seconds...
waiting 9.5 seconds...
waiting 10 seconds...
waiting 10.5 seconds...
waiting 11 seconds...
waiting 11.5 seconds...
waiting 12 seconds...
waiting 12.5 seconds...
waiting 13 seconds...
```

As we see after every 0.5 sec the setInterval function fired and after 3 sec setTimeout function invoked but then it continuously fire the function after every 0.5 sec and not being terminated until we use Ctrl+C.

So next we use clearInterval() function to resolve the above problem as follow

```
timers.js
1  var waitTime = 3000; // 3sec
2  var currentTime = 0;
3  var waitInterval = 500; // 0.5sec
4
5  console.log("wait for it");
6
7  var interval = setInterval(function(){
8
9      currentTime += waitInterval;
10
11      console.log(`waiting ${currentTime/1000}second...`);
12
13  }, waitInterval);
14
15  setTimeout(function(){
16
17      clearInterval(interval);
18      |
19      console.log("done");
20
21  }, waitTime)
22
23
24
25
```

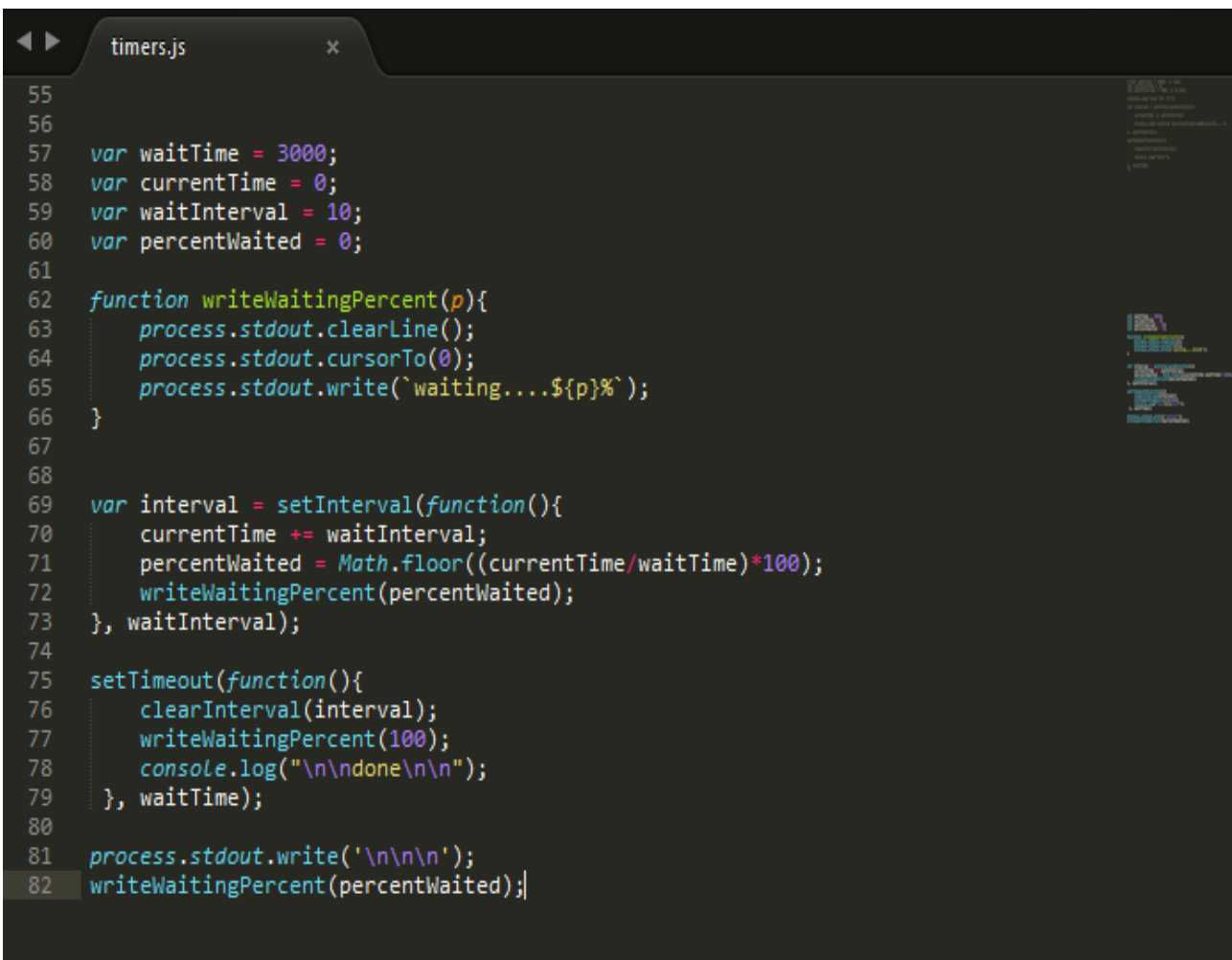
Then go to terminal window and run app as follow “node timers”

Node.js command prompt

```
E:\Exercise Files\Ch03\03_04\start>node timers
wait for it
waiting 0.5second...
waiting 1second...
waiting 1.5second...
waiting 2second...
waiting 2.5second...
done

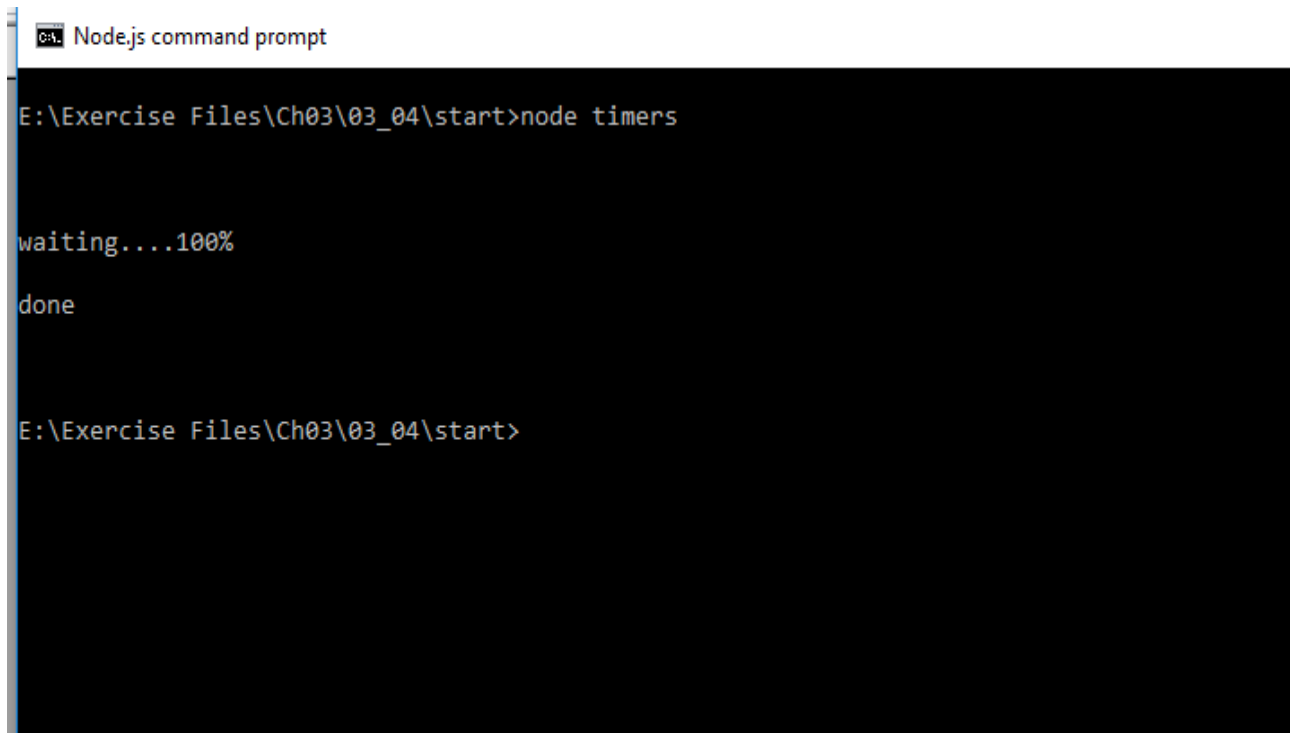
E:\Exercise Files\Ch03\03_04\start>
```

Now we create a function to show waiting in percentage format as follow

A screenshot of a code editor with a dark theme. The editor has a tab titled 'timers.js'. The code is written in JavaScript and implements a timer that updates the progress percentage every 10 milliseconds. It uses `setInterval` to update the progress and `setTimeout` to trigger the final 'done' message. The `writeWaitingPercent` function is used to format and output the progress percentage to the console, clearing the previous line and moving the cursor to the start of the line. The code is as follows:

```
55
56
57 var waitTime = 3000;
58 var currentTime = 0;
59 var waitInterval = 10;
60 var percentWaited = 0;
61
62 function writeWaitingPercent(p){
63     process.stdout.clearLine();
64     process.stdout.cursorTo(0);
65     process.stdout.write(`waiting...${p}%`);
66 }
67
68
69 var interval = setInterval(function(){
70     currentTime += waitInterval;
71     percentWaited = Math.floor((currentTime/waitTime)*100);
72     writeWaitingPercent(percentWaited);
73 }, waitInterval);
74
75 setTimeout(function(){
76     clearInterval(interval);
77     writeWaitingPercent(100);
78     console.log("\ndone\n\n");
79 }, waitTime);
80
81 process.stdout.write('\n\n\n');
82 writeWaitingPercent(percentWaited);
```

Then go to terminal window and run the app as follow

A screenshot of a terminal window titled "Node.js command prompt". The terminal shows the command prompt "E:\Exercise Files\Ch03\03_04\start>node timers". The output of the command is "waiting....100%" followed by "done" on the next line. The prompt "E:\Exercise Files\Ch03\03_04\start>" is shown again at the bottom of the terminal.

```
Node.js command prompt

E:\Exercise Files\Ch03\03_04\start>node timers

waiting....100%
done

E:\Exercise Files\Ch03\03_04\start>
```

Node Modules:-

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Each module in Node.js has its own context, so it cannot interfere with other modules or pollute global scope. Also, each module can be placed in a separate .js file under a separate folder.

Node.js implements [CommonJS modules standard](#). CommonJS is a group of volunteers who define JavaScript standards for web server, desktop, and console application.

Module Types:-

Node.js includes three types of modules:

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

Core Modules:-

Node.js is a light weight framework. The core modules include bare minimum functionalities of Node.js. These core modules are compiled into its binary distribution and load automatically when Node.js process starts. However, you need to import the core module first in order to use it in your application.

Examples:- http, url, path, fs,util etc. are core modules available in nodejs.

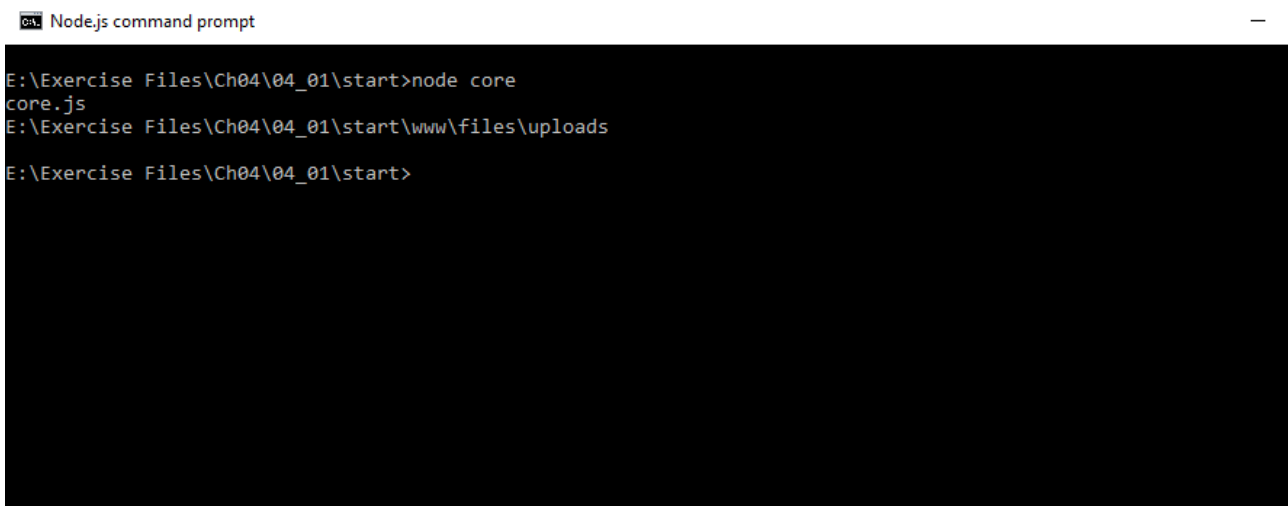
Example:-

Create a folder name start and include a file name core.js then open that file in text editor here we use **path** module with **join()** function to join path further as shown.

A screenshot of a code editor window titled 'core.js'. The code is as follows:

```
1 var path = require('path');
2 //var util = require('util');
3 //var v8 = require('v8');
4
5 console.log(path.basename(__filename));
6
7 var dirUploads = path.join(__dirname, 'www', 'files', 'uploads');
8
9 console.log(dirUploads);
10
11 //util.log(v8.getHeapStatistics());
```

Then go to terminal window and run the app as follow

A screenshot of a terminal window titled 'Node.js command prompt'. The terminal shows the following commands and output:

```
E:\Exercise Files\Ch04\04_01\start>node core
core.js
E:\Exercise Files\Ch04\04_01\start\www\files\uploads
E:\Exercise Files\Ch04\04_01\start>
```

Next we use “**utility module**” named as **util** and it does the same thing as of console but util additionally show date with data on terminal as shown bellow

```
core.js x
1 var path = require('path');
2 var util = require('util');
3 //var v8 = require('v8');
4
5 util.log(path.basename(__filename));
6
7 var dirUploads = path.join(__dirname, 'www', 'files', 'uploads');
8
9 util.log(dirUploads);
10
11 //util.log(v8.getHeapStatistics());
```

Then go to terminal and run the app as follow

```
CA: Node.js command prompt
E:\Exercise Files\Ch04\04_01\start>node core
16 Dec 12:10:57 - core.js
16 Dec 12:10:57 - E:\Exercise Files\Ch04\04_01\start\www\files\uploads
E:\Exercise Files\Ch04\04_01\start>
```

Next we are going to use “**v8 module**” to show memory status as follow
“util.log(v8.getHeapStatistics())”

```
core.js x
1 var path = require('path');
2 var util = require('util');
3 var v8 = require('v8');
4
5 util.log(path.basename(__filename));
6
7 var dirUploads = path.join(__dirname, 'www', 'files', 'uploads');
8
9 util.log(dirUploads);
10
11 util.log(v8.getHeapStatistics());
```

Then go to terminal window and run the app as follow

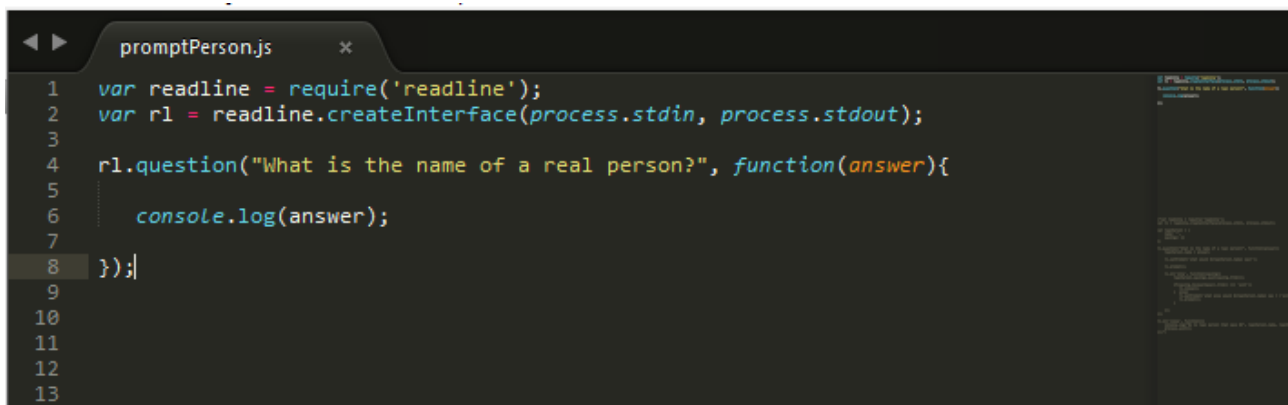
Node.js command prompt

```
E:\Exercise Files\Ch04\04_01\start>node core
16 Dec 12:29:51 - core.js
16 Dec 12:29:51 - E:\Exercise Files\Ch04\04_01\start\www\files\uploads
16 Dec 12:29:51 - { total_heap_size: 10522624,
  total_heap_size_executable: 5242880,
  total_physical_size: 10522624,
  total_available_size: 1488102264,
  used_heap_size: 4310056,
  heap_size_limit: 1501560832 }
E:\Exercise Files\Ch04\04_01\start>
```

Readline:-

Readline is another module to wrap up the stdout and stdin modules to use them easily. Readline control these module for us.

Example:- Create a folder named start and include a file named promptPerson.js and open that in text editor use readline module as follow

A screenshot of a code editor with a dark theme. The editor has a tab at the top labeled 'promptPerson.js'. The code is as follows:

```
1 var readline = require('readline');
2 var rl = readline.createInterface(process.stdin, process.stdout);
3
4 rl.question("What is the name of a real person?", function(answer){
5
6   console.log(answer);
7
8 });|
9
10
11
12
13
```

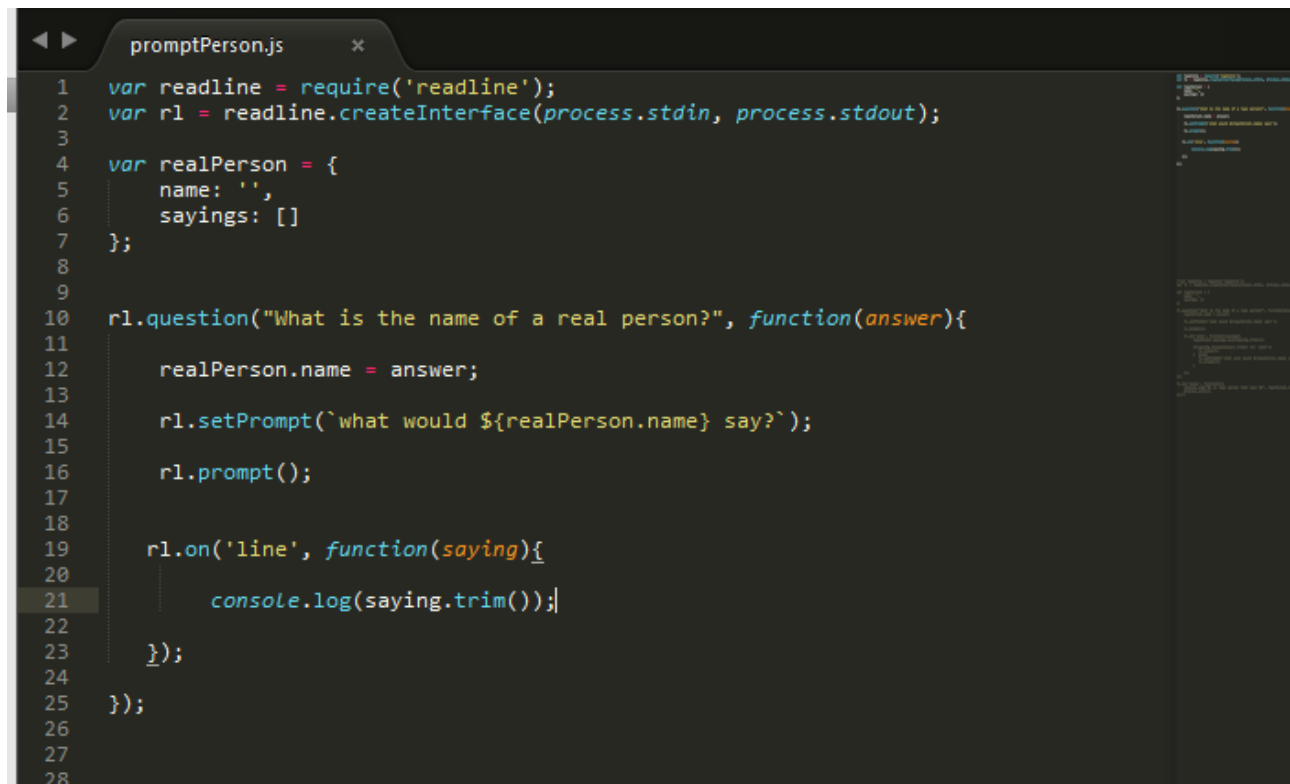
Then go to terminal window and run the app as follow

Node.js command prompt

```
E:\Exercise Files\Ch04\04_02\start>node promptPerson
What is the name of a real person? Swami Vivekananda
Swami Vivekananda

E:\Exercise Files\Ch04\04_02\start>
```

Next in this example we add another functionality that we get information about what real person says as shown.



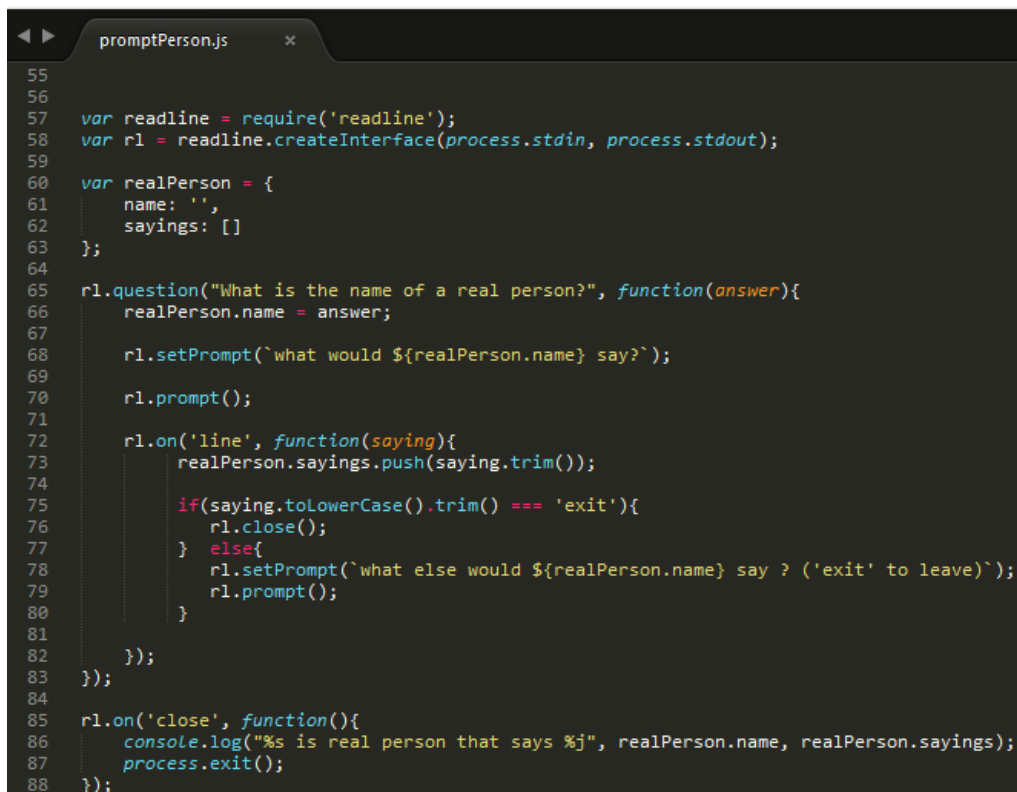
```
promptPerson.js
1 var readline = require('readline');
2 var rl = readline.createInterface(process.stdin, process.stdout);
3
4 var realPerson = {
5   name: '',
6   sayings: []
7 };
8
9
10 rl.question("What is the name of a real person?", function(answer){
11
12   realPerson.name = answer;
13
14   rl.setPrompt(`what would ${realPerson.name} say?`);
15
16   rl.prompt();
17
18   rl.on('line', function(saying){
19
20     console.log(saying.trim());
21
22   });
23
24 });
25
26
27
28
```

Then go to terminal window and run the app

CA: Node.js command prompt - node promptPerson

```
E:\Exercise Files\Ch04\04_02\start>node promptPerson
What is the name of a real person?Swami Vivekananda
What would Swami Vivekananda say?Don't sleep untill you get your goal.
Don't sleep untill you get your goal.
```

Next we use add some code in above file to ask the question again and again until we enter “exit” as follow



```
55
56
57 var readline = require('readline');
58 var rl = readline.createInterface(process.stdin, process.stdout);
59
60 var realPerson = {
61   name: '',
62   sayings: []
63 };
64
65 rl.question("What is the name of a real person?", function(answer){
66   realPerson.name = answer;
67
68   rl.setPrompt(`what would ${realPerson.name} say?`);
69
70   rl.prompt();
71
72   rl.on('line', function(saying){
73     realPerson.sayings.push(saying.trim());
74
75     if(saying.toLowerCase().trim() === 'exit'){
76       rl.close();
77     } else{
78       rl.setPrompt(`what else would ${realPerson.name} say ? ('exit' to leave)`);
79       rl.prompt();
80     }
81   });
82 });
83
84
85 rl.on('close', function(){
86   console.log("%s is real person that says %j", realPerson.name, realPerson.sayings);
87   process.exit();
88 });
```

Then go to terminal window and run the app as follow

```
Node.js command prompt

E:\Exercise Files\Ch04\04_02\start>node promptPerson
What is the name of a real person?Swami Vivekananada
what would Swami Vivekananada say?Dont sleep untill you get your goal
what else would Swami Vivekananada say ? ('exit' to leave)Respect the time, Time will respect you
what else would Swami Vivekananada say ? ('exit' to leave)exit
Swami Vivekananada is real person that says ["Dont sleep untill you get your goal","Respect the time, Time will respect you","exit"]

E:\Exercise Files\Ch04\04_02\start>
```

Event Emitter:-

Event emitter allows us to create an event listener for any custom event. Event emitter is a part of events module so we have to import events module in our file.

Example:- Create a folder name start and include a file name BenFranklin.js and open it in text editor and write code as shown below.

```
BenFranklin.js

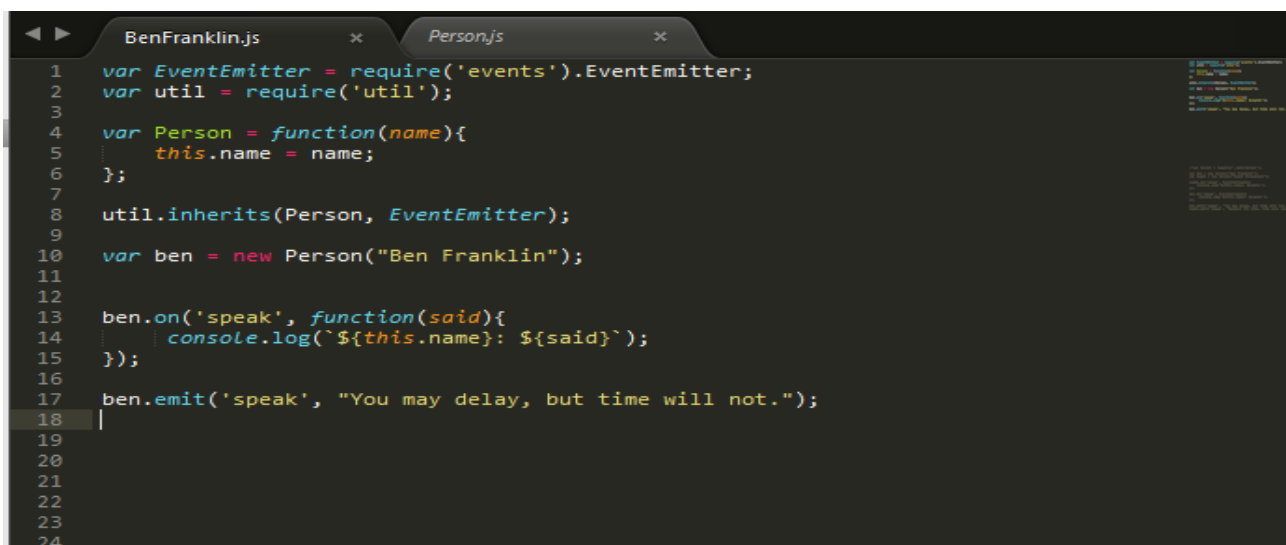
1 var events = require('events');
2
3 var emitter = new events.EventEmitter();
4
5 emitter.on('customEvent', function(message, status){
6   console.log(`${status}: ${message}`);
7 });
8
9 emitter.emit('customEvent', "Hello World", 200);
10
11
12
13
14
15
16
17
18
19
```

Then go to terminal window and run the app as follow

```
E:\Exercise Files\Ch04\04_03\start>node BenFranklin
200: Hello World

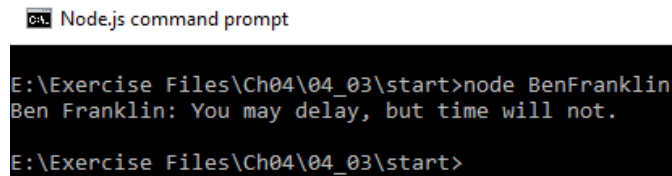
E:\Exercise Files\Ch04\04_03\start>
```

Our next object is to inherit Event Emitter in person object and **util** is module which allows inheritance as shown in the below file



```
BenFranklin.js
1  var EventEmitter = require('events').EventEmitter;
2  var util = require('util');
3
4  var Person = function(name){
5      this.name = name;
6  };
7
8  util.inherits(Person, EventEmitter);
9
10 var ben = new Person("Ben Franklin");
11
12
13 ben.on('speak', function(said){
14     console.log(`${this.name}: ${said}`);
15 });
16
17 ben.emit('speak', "You may delay, but time will not.");
18
19
20
21
22
23
24
```

Then go to terminal and run the app as follow



```
Node.js command prompt
E:\Exercise Files\Ch04\04_03\start>node BenFranklin
Ben Franklin: You may delay, but time will not.
E:\Exercise Files\Ch04\04_03\start>
```

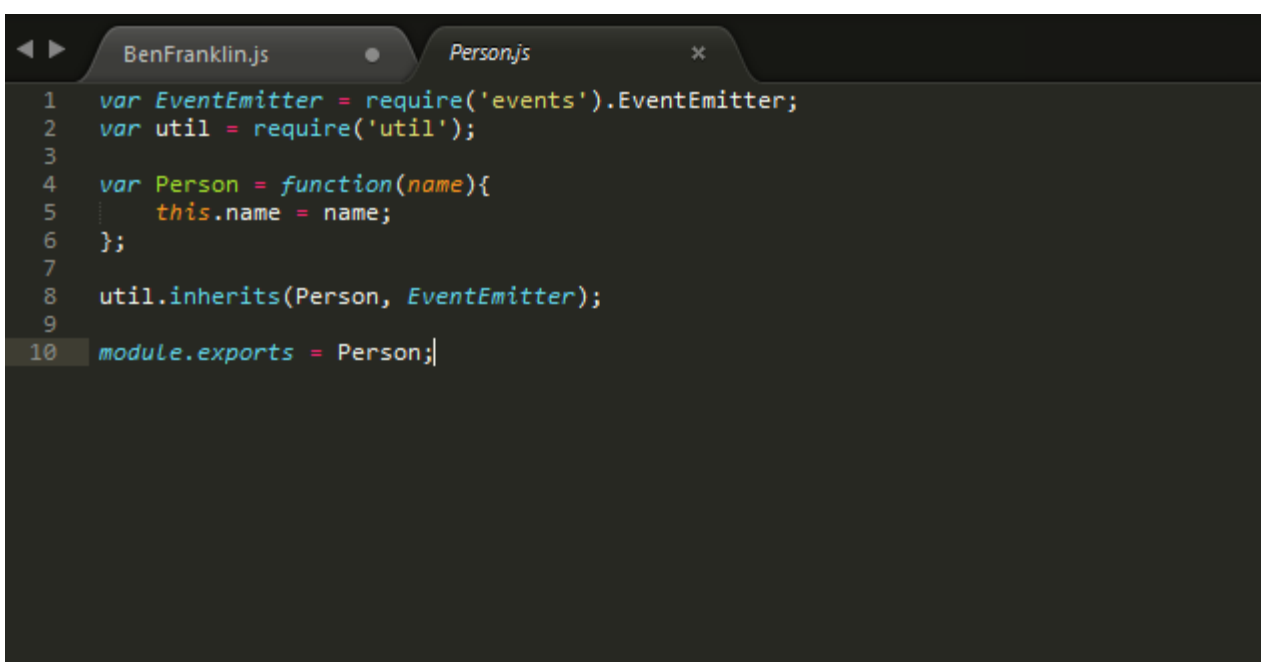
Exporting custom modules:-

Nodejs allows us to export modules so we can use its functionalities anywhere in the application just by importing that module.

Example:- Create a folder named as start and inside the start folder an include a file BenFranklin.js and create a bin folder inside the start folder an include a file person.js as shown

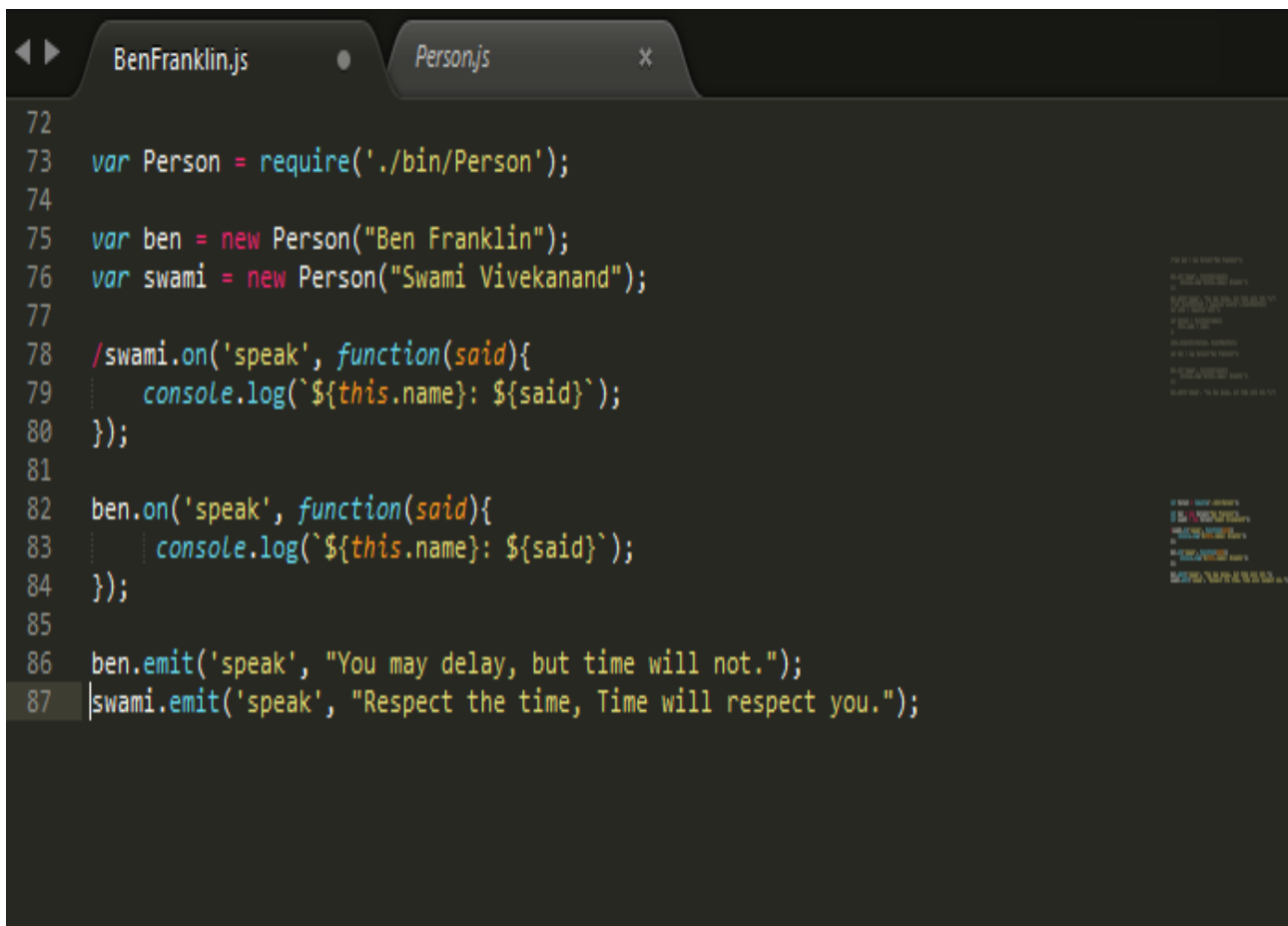
Next create a person.js file as shown. To export a custom module

“module.exports = Person”;



```
BenFranklin.js  Person.js
1  var EventEmitter = require('events').EventEmitter;
2  var util = require('util');
3
4  var Person = function(name){
5    this.name = name;
6  };
7
8  util.inherits(Person, EventEmitter);
9
10 module.exports = Person;
```

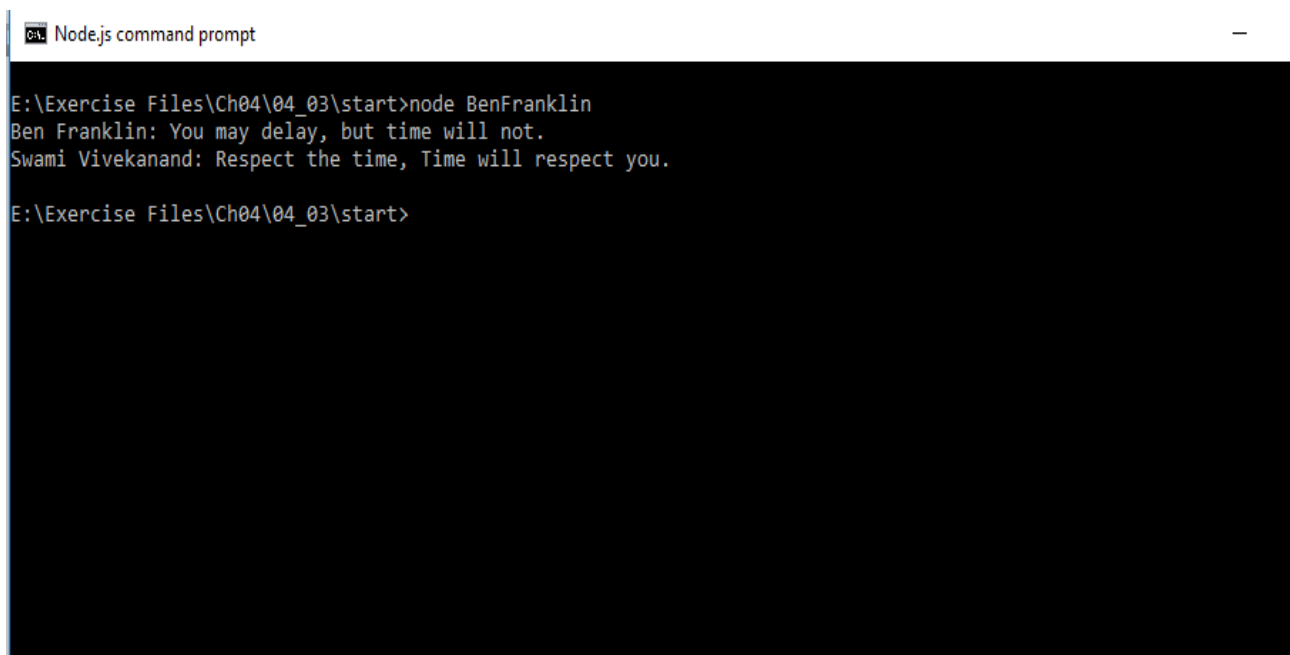
Next create a BenFranklin.js file and import Person module as shown for custom module we have to give path to that file.



The screenshot shows a code editor with two tabs: 'BenFranklin.js' and 'Person.js'. The 'BenFranklin.js' tab is active, displaying the following JavaScript code:

```
72
73 var Person = require('./bin/Person');
74
75 var ben = new Person("Ben Franklin");
76 var swami = new Person("Swami Vivekanand");
77
78 /swami.on('speak', function(said){
79     console.log(`${this.name}: ${said}`);
80 });
81
82 ben.on('speak', function(said){
83     console.log(`${this.name}: ${said}`);
84 });
85
86 ben.emit('speak', "You may delay, but time will not.");
87 swami.emit('speak', "Respect the time, Time will respect you.");
```

Then go to terminal and run the app as follow



The screenshot shows a Node.js command prompt window with the following text:

```
Node.js command prompt
E:\Exercise Files\Ch04\04_03\start>node BenFranklin
Ben Franklin: You may delay, but time will not.
Swami Vivekanand: Respect the time, Time will respect you.
E:\Exercise Files\Ch04\04_03\start>
```

File System:-

Nodejs allows us to interact with file system so its provide “fs” module. Using fs module we can interact with files and directories also. For more on fs module check Nodejs documentation.

Example:- Create a folder name start and create a file list.js and make a bin folder and include people.json, sayings.md and scripts files then open list.js file inside text editor and import “fs” module as shown below.

A screenshot of a code editor with a dark theme. The editor has a tab at the top labeled 'list.js'. The code is written in JavaScript and uses syntax highlighting. The code is as follows:

```
1 |var fs = require("fs");
2
3 |fs.readdir('./lib', function(err, files){
4 |    if(err){
5 |        throw err;
6 |    }
7
8 |    console.log(files);
9 |});
10
11 |console.log("Reading files.....");
```

Then go to terminal and run the app as follow

```
E:\Exercise Files\Ch05\05_01\start>node list
Reading files.....
[ 'people.json', 'sayings.md', 'scripts' ]

E:\Exercise Files\Ch05\05_01\start>
```

Reading files:-

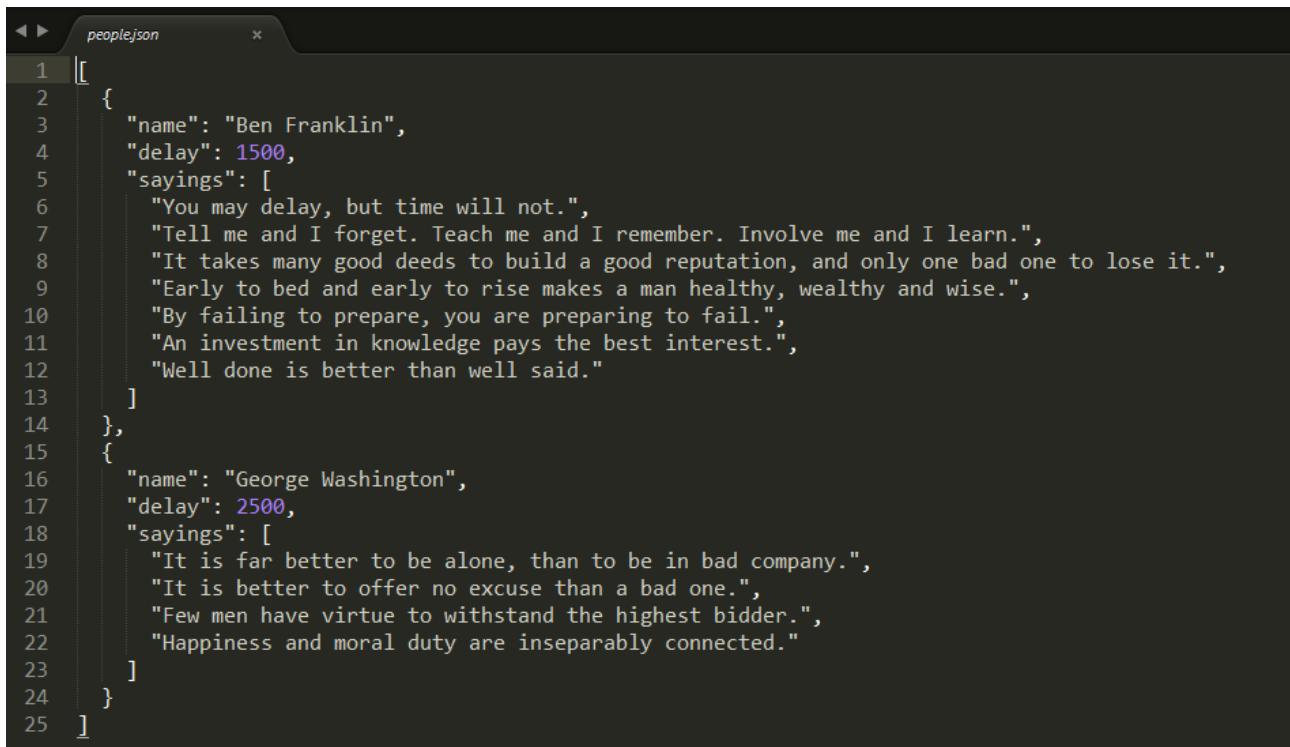
Another feature of “fs” module is read the contains of file. Here is an example to show how

to read contains of files.

Example:-

Create a folder start and include a file named read.js, create a lib folder inside a start folder containing people.json, and sayings.md files .

Create a people.json file as shown below,

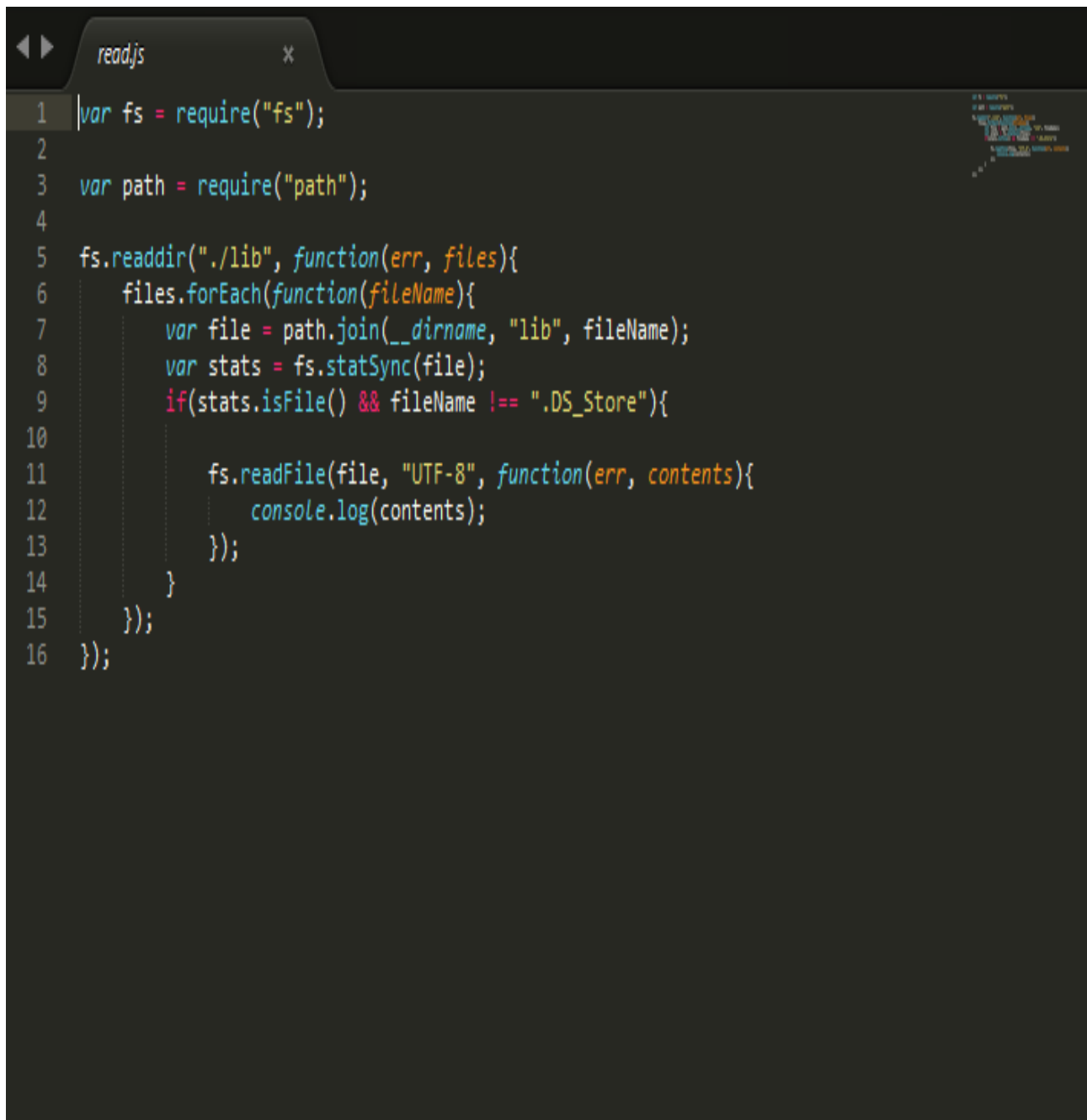
A screenshot of a code editor window titled 'people.json'. The editor shows a JSON array with two objects. The first object represents Ben Franklin with a delay of 1500 and six sayings. The second object represents George Washington with a delay of 2500 and four sayings. The code is syntax-highlighted with colors for strings, numbers, and brackets. Line numbers 1 through 25 are visible on the left side of the editor.

```
1  [
2  {
3    "name": "Ben Franklin",
4    "delay": 1500,
5    "sayings": [
6      "You may delay, but time will not.",
7      "Tell me and I forget. Teach me and I remember. Involve me and I learn.",
8      "It takes many good deeds to build a good reputation, and only one bad one to lose it.",
9      "Early to bed and early to rise makes a man healthy, wealthy and wise.",
10     "By failing to prepare, you are preparing to fail.",
11     "An investment in knowledge pays the best interest.",
12     "Well done is better than well said."
13   ]
14 },
15 {
16   "name": "George Washington",
17   "delay": 2500,
18   "sayings": [
19     "It is far better to be alone, than to be in bad company.",
20     "It is better to offer no excuse than a bad one.",
21     "Few men have virtue to withstand the highest bidder.",
22     "Happiness and moral duty are inseparably connected."
23   ]
24 }
25 ]
```

Also create a sayings.md file as shown below,

```
sayings.md x
1 Ben Franklin
2 =====
3
4 * You may delay, but time will not.
5 * Tell me and I forget. Teach me and I remember. Involve me and I learn.
6 * It takes many good deeds to build a good reputation, and only one bad one to lose it.
7 * Early to bed and early to rise makes a man healthy, wealthy and wise.
8 * By failing to prepare, you are preparing to fail.
9 * An investment in knowledge pays the best interest.
10 * Well done is better than well said.
11
12 George Washington
13 =====
14
15 * It is far better to be alone, than to be in bad company.
16 * It is better to offer no excuse than a bad one.
17 * Few men have virtue to withstand the highest bidder.
18 * Happiness and moral duty are inseparably connected.
19
```

And also create a read.js file as shown below.



```
1 |var fs = require("fs");
2
3 |var path = require("path");
4
5 |fs.readdir("./lib", function(err, files){
6 |    files.forEach(function(fileName){
7 |        var file = path.join(__dirname, "lib", fileName);
8 |        var stats = fs.statSync(file);
9 |        if(stats.isFile() && fileName !== ".DS_Store"){
10 |
11 |            fs.readFile(file, "UTF-8", function(err, contents){
12 |                console.log(contents);
13 |            });
14 |        }
15 |    });
16 |});
```

Then go to terminal window and run the app

```

E:\Exercise Files\Ch05\05_02\start>node read
[
  {
    "name": "Ben Franklin",
    "delay": 1500,
    "sayings": [
      "You may delay, but time will not.",
      "Tell me and I forget. Teach me and I remember. Involve me and I learn.",
      "It takes many good deeds to build a good reputation, and only one bad one to lose it.",
      "Early to bed and early to rise makes a man healthy, wealthy and wise.",
      "By failing to prepare, you are preparing to fail.",
      "An investment in knowledge pays the best interest.",
      "Well done is better than well said."
    ]
  },
  {
    "name": "George Washington",
    "delay": 2500,
    "sayings": [
      "It is far better to be alone, than to be in bad company.",
      "It is better to offer no excuse than a bad one.",
      "Few men have virtue to withstand the highest bidder.",
      "Happiness and moral duty are inseparably connected."
    ]
  }
]
Ben Franklin
=====

* You may delay, but time will not.
* Tell me and I forget. Teach me and I remember. Involve me and I learn.
* It takes many good deeds to build a good reputation, and only one bad one to lose it.
* Early to bed and early to rise makes a man healthy, wealthy and wise.
* By failing to prepare, you are preparing to fail.
* An investment in knowledge pays the best interest.
* Well done is better than well said.

George Washington
=====

* It is far better to be alone, than to be in bad company.
* It is better to offer no excuse than a bad one.

```

Writing and appending the files:-

There is an another feature of writing and appending files in “fs” module.

Example:- Create a folder named start and include two files create.js and promptPerson.js then open the create.js in text editor and write code for create a md file as shown bellow

A screenshot of a code editor with a dark theme. The editor has a tab at the top labeled 'create.js' with a close button (X) on the right. The code is written in JavaScript and is as follows:

```
1 var fs = require('fs');
2
3 var md = `
4     Sample Markdown Title
5     =====
6
7
8
9     Sample subtitle
10    -----
11
12    * point
13    * point
14    * point
15
16 `;
17
18 fs.writeFile("sample.md", md.trim(), function(err){
19     console.log("File Created");
20 });
```

The code is line-numbered from 1 to 20 on the left side of the editor. The file content is a JavaScript script that uses the 'fs' module to write a markdown file named 'sample.md'. The markdown content is stored in a template literal variable 'md' and includes a title, subtitle, and a list of three points. The script uses 'md.trim()' to remove leading and trailing whitespace before writing the file. A callback function logs 'File Created' to the console upon successful completion.

Then go to terminal window and run the app as shown

```
Node.js command prompt
Your environment has been set up for using Node.js 6.9.1 (x64) and npm.

C:\Users\Tek2>e:

E:\>cd "Exercise Files"

E:\Exercise Files>cd Ch05

E:\Exercise Files\Ch05>cd 05_03

E:\Exercise Files\Ch05\05_03>cd start

E:\Exercise Files\Ch05\05_03\start>node create
File Created

E:\Exercise Files\Ch05\05_03\start>
```

As we see “**File created**” shown on console screen so a file named sample.md created in start folder.

Local Disk (E:) > Exercise Files > Ch05 > 05_03 > start				Search start	
Name		Date modified		Type	
	create.js	16/12/2016 15:44		JavaScript File	
	promptPerson.js	07/12/2016 16:09		JavaScript File	
	sample.md	16/12/2016 15:46		MD File	
	swami vivekanand.md	07/12/2016 16:10		MD File	

Next for writing and appending the file open the promptPerson.js in text editor and write codes as shown below

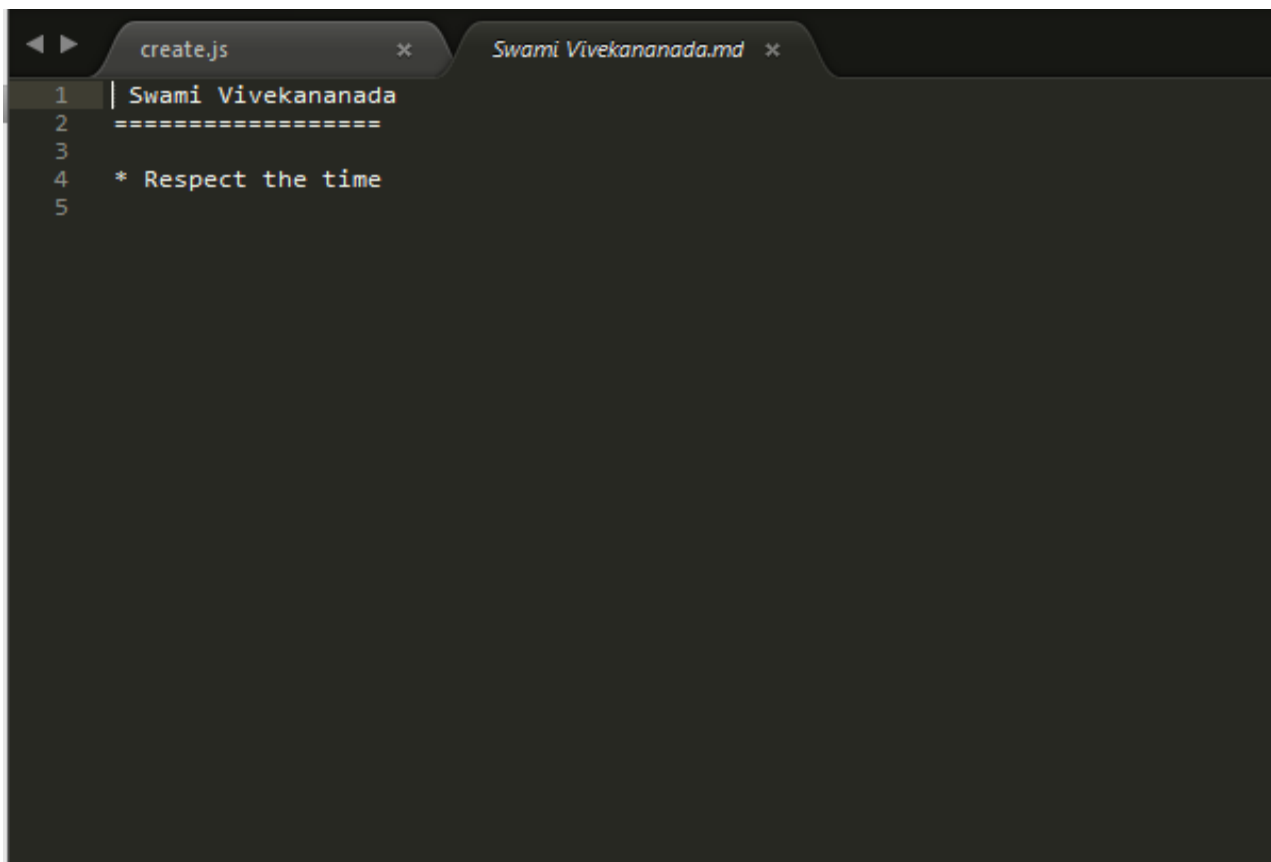
```
create.js x promptPerson.js x
1 |var readline = require('readline');
2 |var rl = readline.createInterface(process.stdin, process.stdout);
3 |var fs = require('fs');
4 |
5 |var realPerson = {
6 |  name: '',
7 |  sayings: []
8 |};
9 |
10 |rl.question("What is the name of a real person?", function(answer){
11 |  realPerson.name = answer;
12 |
13 |  fs.writeFileSync(realPerson.name + ".md", `${realPerson.name}\n=====\\n\\n`);
14 |
15 |  rl.setPrompt(`what would ${realPerson.name} say?`);
16 |
17 |  rl.prompt();
18 |
19 |  rl.on('line', function(saying){
20 |    realPerson.sayings.push(saying.trim());
21 |
22 |    fs.appendFile(realPerson.name + ".md", `* ${saying.trim()}\\n`);
23 |
24 |    if(saying.toLowerCase().trim() === 'exit'){
25 |      rl.close();
26 |    } else{
27 |      rl.setPrompt(`what else would ${realPerson.name} say ? ('exit' to leave)`);
28 |      rl.prompt();
29 |    }
30 |
31 |  });
32 |});
33 |
34 |rl.on('close', function(){
35 |  console.log("%s is real person that says %j", realPerson.name, realPerson.sayings);
36 |  process.exit();
37 |});
```

Then run the app using “**node promptPerson**”

Node.js command prompt

```
E:\Exercise Files\Ch05\05_03\start>promptPerson  
  
E:\Exercise Files\Ch05\05_03\start>node promptPerson  
What is the name of a real person? Swami Vivekananada  
What would Swami Vivekananada say? Respect the time  
What else would Swami Vivekananada say ? ('exit' to leave) exit  
Swami Vivekananada is real person that says ["Respect the time","exit"]  
  
E:\Exercise Files\Ch05\05_03\start>
```

Now go to your folder a new file name Swami Vivekananada.md created and also appended



```
1 | Swami Vivekananada  
2 | =====  
3 |  
4 | * Respect the time  
5 |
```

Directory creation:-

“fs” module gives a feature not only work with file but also with directories. Here is an


example to show “fs” module create directory.

Example:- Create a start folder include file named directory.js open it in the text editor write code as shown below

A screenshot of a code editor with a dark theme. The editor has a tab at the top labeled 'directory.js' with a close button (X) on the right. The code is written in JavaScript and is as follows:

```
1 |var fs = require('fs');
2
3 |if(fs.existsSync("lib")){
4 |    console.log("directory already exist");
5 |}
6
7 |else{
8 |    fs.mkdir("lib", function(err){
9 |        if(err){
10 |            console.log(err);
11 |        } else{
12 |            console.log("directory created");
13 |        }
14 |    });
15 |}
```


Then run the app using “node directory”

 Node.js command prompt

```
E:\Exercise Files\Ch05\05_04\start>node directory  
directory created
```


```
E:\Exercise Files\Ch05\05_04\start>
```


Now go to your folder new directory named **lib** will be created

 E:\Exercise Files\Ch05\05_04\start\directory.js (start) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

▼  start

▶  lib

 directory.js



directory.js



```
1 |var fs = require('fs');
2
3 if(fs.existsSync("lib")){
4     console.log("directory already exist");
5 }
6
7 else{
8     fs.mkdir("lib", function(err){
9         if(err){
10             console.log(err);
11         } else{
12             console.log("directory created");
13         }
14     });
15 }
```

Renaming and Removing files:-

There are another feature renaming and removing files also available in “fs” module.

Create a folder name start and create files name remove.js and rename.js and a folder lib. Lib folder also contains two files name project-config.js and note.md.


Open rename.js file in text editor and write code shown below

A screenshot of a code editor with a dark theme. At the top, there are two tabs: 'rename.js' and 'remove.js'. The 'rename.js' tab is active, showing the following JavaScript code:

```
1 |var fs = require("fs");
2
3 fs.renameSync("./lib/project-config.js", "./lib/config.json");
4
5 console.log("config json file renamed");
6
7 fs.rename("./lib/notes.md", "./notes.md", function(err){
8     if(err){
9         console.log(err);
10    } else{
11        console.log("Notes.md moved succesfully");
12    }
13 });
```

Then go to terminal and run the app using

” node rename”

 Node.js command prompt

```
E:\Exercise Files\Ch05\05_05\start>node rename  
config json file renamed  
Notes.md moved succesfully  
  
E:\Exercise Files\Ch05\05_05\start>
```

Now check the folder project-config.js renamed as config.json and notes.md moved as shown

E:\Exercise Files\Ch05\05_05\start\rename.js (start) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

- start
 - lib
 - config.json
 - notes.md
 - remove.js
 - rename.js

```
1 |var fs = require("fs");
2
3 fs.renameSync("./lib/project-config.js"
4
5 console.log("config json file renamed")
6
7 fs.rename("./lib/notes.md", "../notes.md"
8     if(err){
9         console.log(err);
10    } else{
11        console.log("Notes.md moved suc
12    }
13 });
```


Now open the remove.js file in text editor and write codes as shown below

```
1 |var fs = require("fs");
2
3 try{
4     fs.unlinkSync("./lib/config.json");
5 } catch(err) {
6     console.log(err);
7 }
8
9 fs.unlink("notes.md", function(err){
10     if(err){
11         console.log(err);
12     } else{
13         console.log("notes.md removed")
14     }
15 })
```

Go to terminal window and run app using “node remove”


```
E:\Exercise Files\Ch05\05_05\start>node rename  
config json file renamed  
Notes.md moved succesfully  
  
E:\Exercise Files\Ch05\05_05\start>node remove  
notes.md removed  
  
E:\Exercise Files\Ch05\05_05\start>
```


In the start folder notes.md file removed

 E:\Exercise Files\Ch05\05_05\start\remove.js (start) - Sublime Text (UNREGISTERED)


File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

▼  start

▼  lib

 remove.js

 rename.js

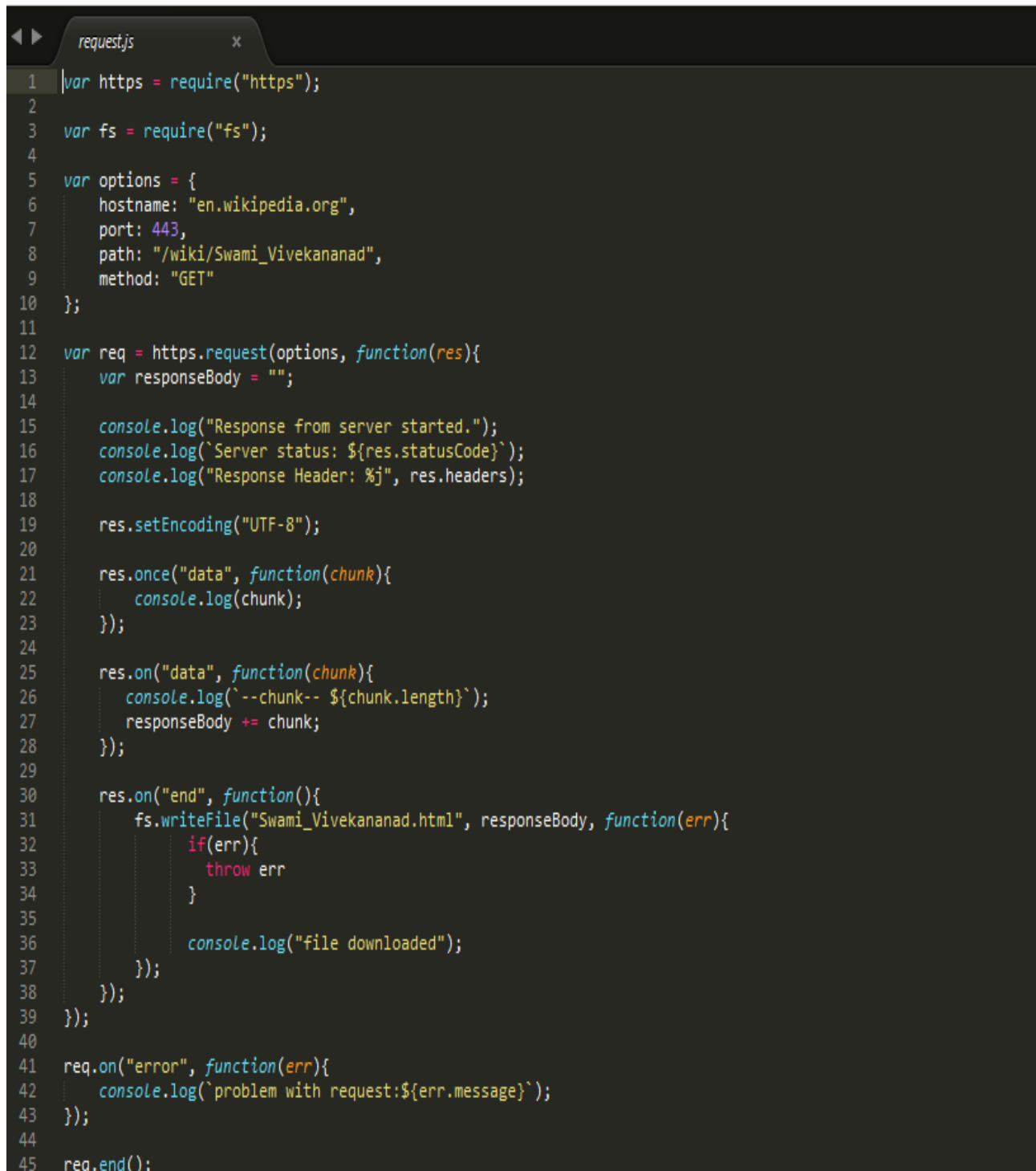
```
1 |var fs = require("fs");
2
3 |try{
4 |    fs.unlinkSync("./lib/config.json");
5 |} catch(err) {
6 |    console.log(err);
7 |}
8
9 |fs.unlink("notes.md", function(err){
10 |    if(err){
11 |        console.log(err);
12 |    } else{
13 |        console.log("notes.md removed")
14 |    }
15 |})
```


The HTTP module:-

HTTP module is used to creating web server, making request, handling responses etc. There are two modules http and https, both are very similar but we work with https when we use secure server. For more on http module check nodejs documentation.

Making a request:-

Example:- Create a folder named start and include a file name request.js. Open that in the text editor and write the codes as shown below



```
1 |var https = require("https");
2
3 |var fs = require("fs");
4
5 |var options = {
6 |    hostname: "en.wikipedia.org",
7 |    port: 443,
8 |    path: "/wiki/Swami_Vivekananad",
9 |    method: "GET"
10|};
11
12|var req = https.request(options, function(res){
13|    var responseBody = "";
14
15|    console.log("Response from server started.");
16|    console.log(`Server status: ${res.statusCode}`);
17|    console.log("Response Header: %j", res.headers);
18
19|    res.setEncoding("UTF-8");
20
21|    res.once("data", function(chunk){
22|        console.log(chunk);
23|    });
24
25|    res.on("data", function(chunk){
26|        console.log(`--chunk-- ${chunk.length}`);
27|        responseBody += chunk;
28|    });
29
30|    res.on("end", function(){
31|        fs.writeFile("Swami_Vivekananad.html", responseBody, function(err){
32|            if(err){
33|                throw err
34|            }
35
36|            console.log("file downloaded");
37|        });
38|    });
39|});
40
41|req.on("error", function(err){
42|    console.log(`problem with request:${err.message}`);
43|});
44
45|req.end();
```

Go to terminal window and run the app using” node request”;

A Swami_Vivekananad html page will be downloaded on http.request.

```
Nodejs command prompt
<title>Swami Vivekananad - Wikipedia</title>
<script>document.documentElement.className = document.documentElement.className.replace( /(
--chunk-- 239
--chunk-- 1300
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 1300
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 1300
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 1300
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 1300
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 1298
--chunk-- 1300
--chunk-- 196
--chunk-- 1300
--chunk-- 922
File downloaded
```

Now navigate to folder a html page downloaded as shown

```
E:\Exercise Files\Ch06\06_01\start\request.js (start) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

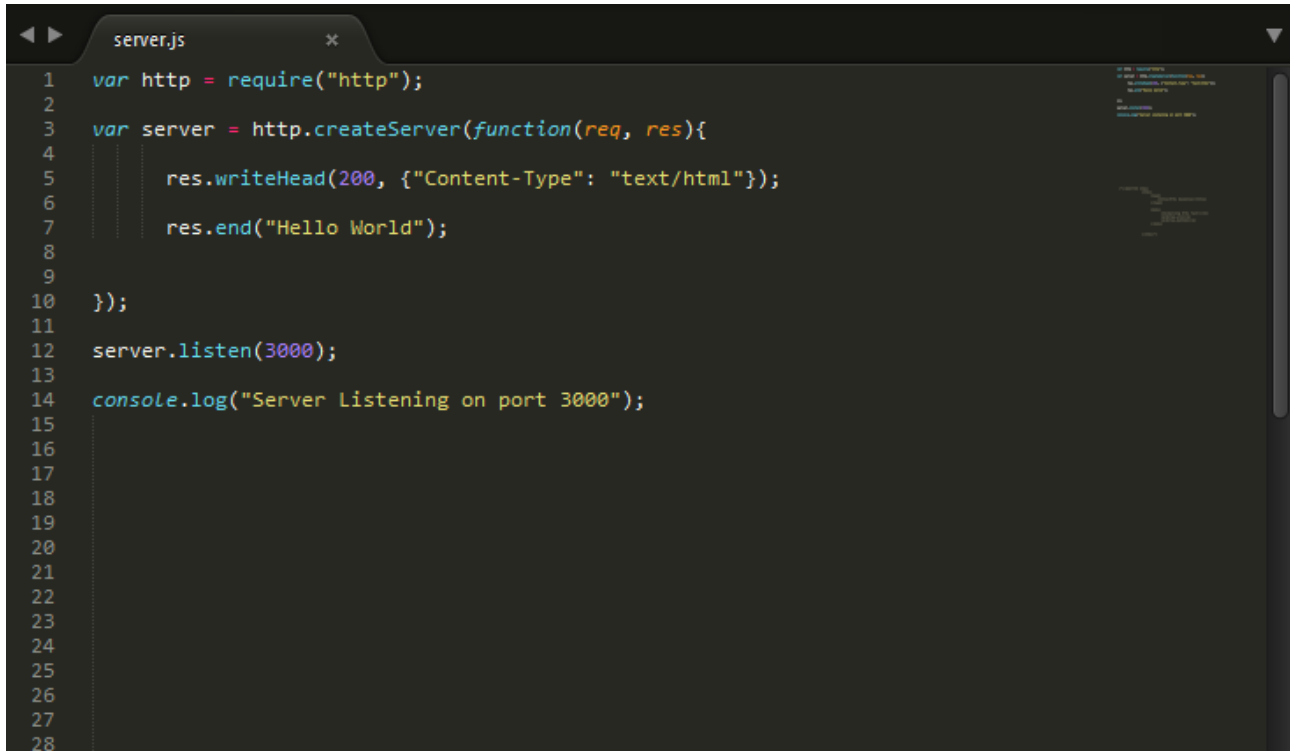
FOLDERS
▼ start
  request.js
  Swami_Vivekananad.html

request.js
1 |var https = require("https");
2
3   var fs = require("fs");
4
5   var options = {
6     hostname: "en.wikipedia.org",
7     port: 443,
8     path: "/wiki/Swami_Vivekana",
9     method: "GET"
10  };
11
12  var req = https.request(options, function(res) {
13    var responseBody = "";
14
15    console.log("Response from");
16    console.log(`Server status: ${res.statusCode}`);
17    console.log("Response Headers:");
18    for (var header in res.headers) {
19      console.log(`${header}: ${res.headers[header]}`);
20    }
21    res.setEncoding("UTF-8");
22    res.on("data", function(chunk) {
23      console.log(chunk);
24    });
25    res.on("end", function() {
26      console.log("Request completed");
27    });
28  });
29  req.end();
```

Building Web Server:-

HTTP module in Nodejs allows us to create a web server we can use both http or https module to create a web server. Here is an example how to create web server in node js.

Example:- Create a folder named start and include a file name server.js open up that file in text editor and write code shown below.



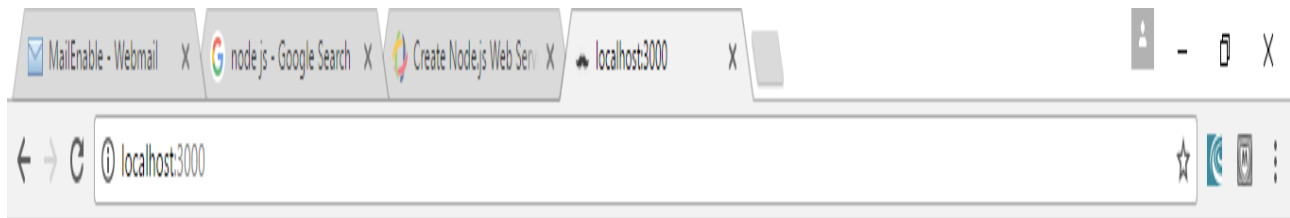
```
1  var http = require("http");
2
3  var server = http.createServer(function(req, res){
4      res.writeHead(200, {"Content-Type": "text/html"});
5      res.end("Hello World");
6  });
7
8  server.listen(3000);
9
10 console.log("Server Listening on port 3000");
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
```

Then run the app using “node server”

Node.js command prompt - node server

```
E:\Exercise Files\Ch06\06_02\start>node server  
Server Listening on port 3000
```

Server listening on localhost3000 so go to browser open <http://localhost3000>



Hello World

Serving the files:-

HTTP module also having feature to serve the files like html, css, image files etc. Here is an example showing how http module is used to serve the files

Example:- Create a folder named start include any index.html file in public folder and a file named fileserver.js in root folder(start) open up that file in the text editor and write code shown below..

```
fileserv.js x
1 var http = require("http");
2
3 var fs = require("fs");
4 var path = require("path");
5
6 http.createServer(function(req, res){
7     console.log(`${req.method} request for ${req.url}`);
8
9     if(req.url === "/") {
10         fs.readFile("./public/index.html", "UTF-8", function(err, html){
11             res.writeHead(200, {"Content-Type": "text/html"});
12             res.end(html);
13         });
14     }
15
16     else if(req.url.match(/.css$/)){
17
18         var cssPath = path.join(__dirname, 'public', req.url);
19         var fileStream = fs.createReadStream(cssPath, "UTF-8");
20
21         res.writeHead(200, {"Content-Type": "text/css"});
22         fileStream.pipe(res);
23     }
24
25     else if(req.url.match(/.jpg$/)){
26
27         var imgPath = path.join(__dirname, 'public', req.url);
28         var imgStream = fs.createReadStream(imgPath);
29
30         res.writeHead(200, {"Content-Type": "image/jpeg"});
31         imgStream.pipe(res);
32     }
33
34     else{
35         res.writeHead(404, {"Content-Type": "text/plain"});
36         res.end("404 File Not Found");
37     }
38
39 }).listen(3000);
```

Then run the app using “node fileserv”

```
Node.js command prompt - node fileserver

E:\Exercise Files\Ch06\06_02\start>node server
Server Listening on port 3000
^C
E:\Exercise Files\Ch06\06_02\start>cd /

E:\>cd "Exercise Files"

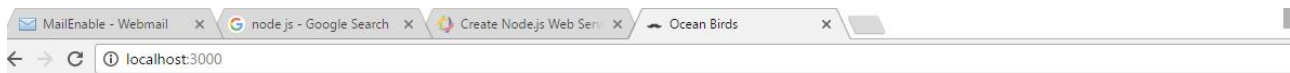
E:\Exercise Files>CD Ch06

E:\Exercise Files\Ch06>cd 06_03

E:\Exercise Files\Ch06\06_03>cd start

E:\Exercise Files\Ch06\06_03\start>node fileserver
GET request for /
GET request for /birds.jpg
GET request for /style.css
GET request for /favicon.ico
```

Open the browser and go to <http://localhost:3000>



Ocean Birds




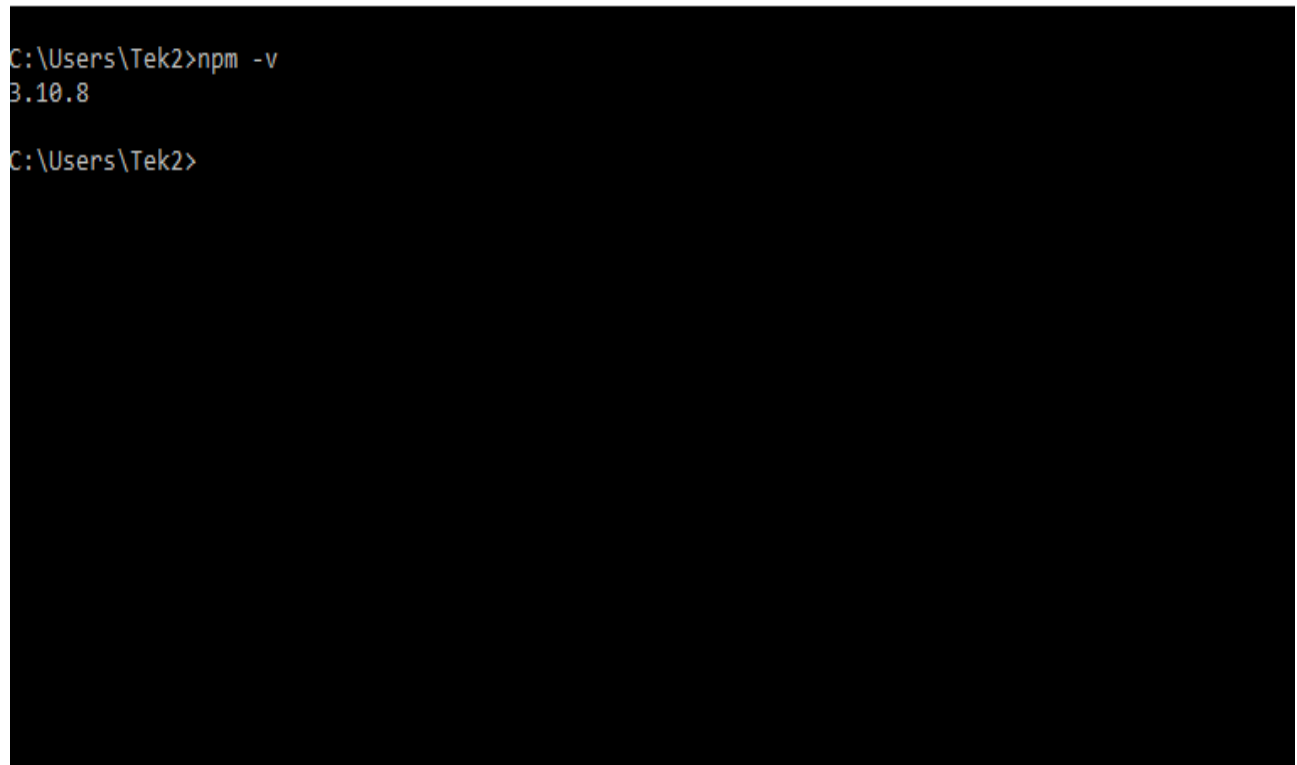
Birds flying over the ocean with the marine layer in the horizon.

Node Package Manager:-

Node js cores are relatively small it only contain essential modules so Node js installation come with a powerful tool Node Package Manager(NPM). This is source which allow us to install modules created by open source community.

To check the version of npm go to command prompt and type “npm -v” it shows version installed on your system

 Node.js command prompt



```
C:\Users\Tek2>npm -v
3.10.8
C:\Users\Tek2>
```

Installing npm locally:-

To install package locally just go to command prompt open your folder where you want to install package locally then type following command

“npm install underscore”

This command install underscore module locally on your system


```
Node.js command prompt

C:\>e:

E:\>cd "New folder"

E:\New folder>cd 1

E:\New folder\1>npm install underscore
E:\New folder\1
-- underscore@1.8.3

npm WARN enoent ENOENT: no such file or directory, open 'E:\New folder\1\package.json'
npm WARN 1 No description
npm WARN 1 No repository field.
npm WARN 1 No README data
npm WARN 1 No license field.

E:\New folder\1>
```

Installing npm globally on pc:-

To install packages globally on pc we need to have administrator privileges. So search for cmd on your pc and open it up as a administrator then go to your folder then type code as shown

“npm install -g node-dev”

Here -g stand for globally. The above code install node-dev module globally.

```
Node.js command prompt

E:\New folder\1>npm install -g node-dev
npm WARN deprecated object-keys@0.2.0: Please update to the latest object-keys
C:\Users\Tek2\AppData\Roaming\npm\node-dev -> C:\Users\Tek2\AppData\Roaming\npm\node_modules\node-dev\bin\node-dev
C:\Users\Tek2\AppData\Roaming\npm
-- node-dev@3.1.3
  -- resolve@1.2.0

E:\New folder\1>
```

Web Server :-

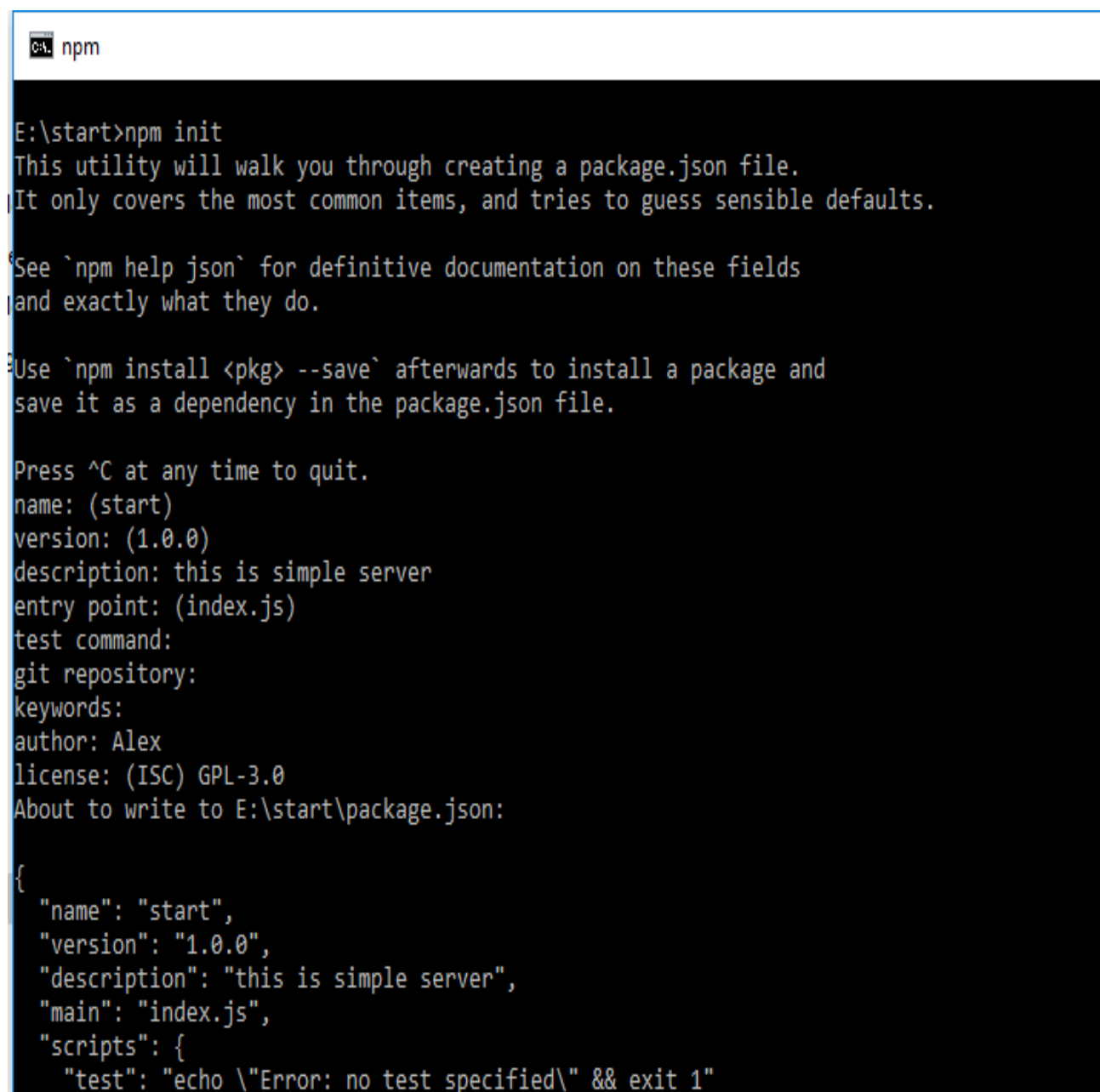
To do more than just serve the static files we require some more powerful tool like express. Express is very popular framework for developing web server application. Express is very popular framework for Nodejs.

The package.json file:-

All npm packages contain a **file**, usually in the project root, called **package.json** this **file** holds various metadata relevant to the project. This **file** is used to give information to npm that allows it to identify the project as well as handle the project's dependencies.

To create a package json file in a empty folder named start then go to terminal window and type code as shown

“npm init”



```
C:\ npm

E:\start>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.


See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

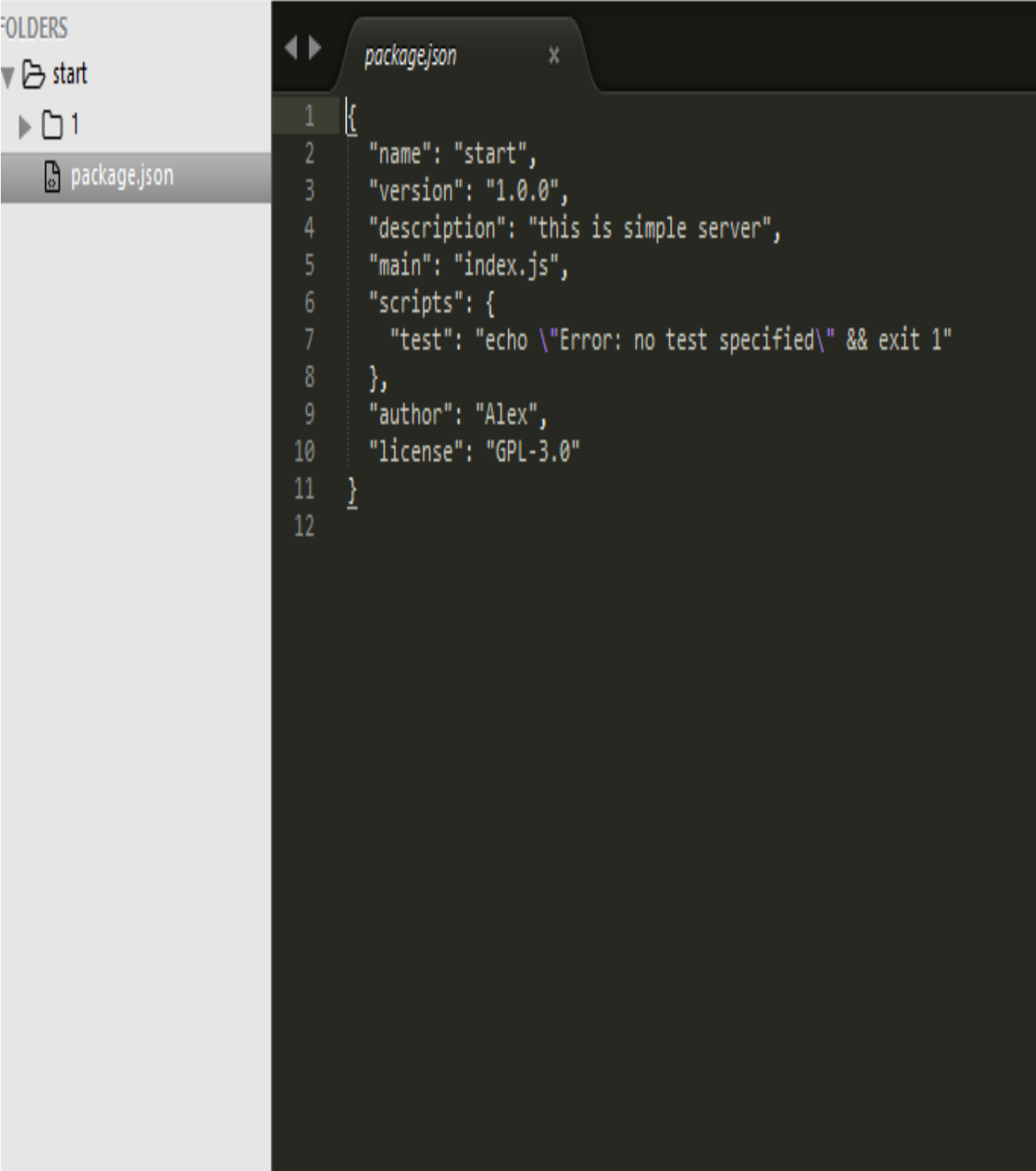
Press ^C at any time to quit.
name: (start)
version: (1.0.0)
description: this is simple server
entry point: (index.js)
test command:
git repository:
keywords:
author: Alex
license: (ISC) GPL-3.0
About to write to E:\start\package.json:

{
  "name": "start",
  "version": "1.0.0",
  "description": "this is simple server",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  }
}
```

It will create a package.json file as shown below

 E:\start\package.json (start) - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help



```
1 {  
2   "name": "start",  
3   "version": "1.0.0",  
4   "description": "this is simple server",  
5   "main": "index.js",  
6   "scripts": {  
7     "test": "echo \"Error: no test specified\" && exit 1"  
8   },  
9   "author": "Alex",  
10  "license": "GPL-3.0"  
11 }  
12
```

Next step is go to terminal open your file and install express as shown

“npm install express –save”

Here save flag add this dependencies to package.json file.

1 npm install express --save

File Edit Selection Find View Goto Tools Project Preferences Help

FOLDERS

start

1

node_modules

package.json

package.json

```
1 {
2   "name": "start",
3   "version": "1.0.0",
4   "description": "this is simple server",
5   "main": "index.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "author": "Alex",
10  "license": "GPL-3.0",
11  "dependencies": {
12    "express": "^4.14.0"
13  }
14 }
15
```

```
E:\start>npm install express --save
start@1.0.0 E:\start
`-- express@4.14.0
   +-- accepts@1.3.3
   | +-- mime-types@2.1.13
   | | `-- mime-db@1.25.0
   | `-- negotiator@0.6.1
   +-- array-flatten@1.1.1
   +-- content-disposition@0.5.1
   +-- content-type@1.0.2
   +-- cookie@0.3.1
   +-- cookie-signature@1.0.6
   +-- debug@2.2.0
   | `-- ms@0.7.1
   +-- depd@1.1.0
   +-- encodeurl@1.0.1
   +-- escape-html@1.0.3
   +-- etag@1.7.0
   +-- finalhandler@0.5.0
   | +-- statuses@1.3.1
   | `-- unpipe@1.0.0
   +-- fresh@0.3.0
   +-- merge-descriptors@1.0.1
   +-- methods@1.1.2
   +-- on-finished@2.3.0
   | `-- ee-first@1.1.1
   +-- parseurl@1.3.1
   +-- path-to-regexp@0.1.7
   +-- proxy-addr@1.1.2
   | +-- forwarded@0.1.0
   | `-- ipaddr.js@1.1.1
   +-- qs@6.2.0
   +-- range-parser@1.2.0
   +-- send@0.14.1
   | +-- destroy@1.0.4
   | +-- http-errors@1.5.1
   | | +-- inherits@2.0.3
   | | `-- setprototypeof@1.0.2
   | `-- mime@1.3.4
   +-- serve-static@1.11.1
   +-- type-is@1.6.14
   | `-- media-typer@0.3.0
   +-- utils-merge@1.0.0
```

Introduction to Express:-

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

Installing Express :-

Assuming we have already installed Nodejs then first step is create a new directory as following

“mkdir myApp”

Or you can manually create a directory in your folder.

Create a folder start and open it in terminal window and type

“mkdir myApp”

This will create a directory in start folder

Node.js command prompt

```
E:\start>mkdir myApp
E:\start>cd myApp
E:\start\myApp>
```

Next step is create a package.json file using “npm init” command so go to terminal window and type

“ npm init”

npm

```
E:\start\myApp>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg> --save` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
name: (myapp) myapp
version: (1.0.0)
git repository:
keywords:
license: (GPL-3.0)
About to write to E:\start\myApp\package.json:
{
  "name": "myapp",
  "version": "1.0.0",
  "description": "this is simple app",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Alex",
  "license": "GPL-3.0"
}
```

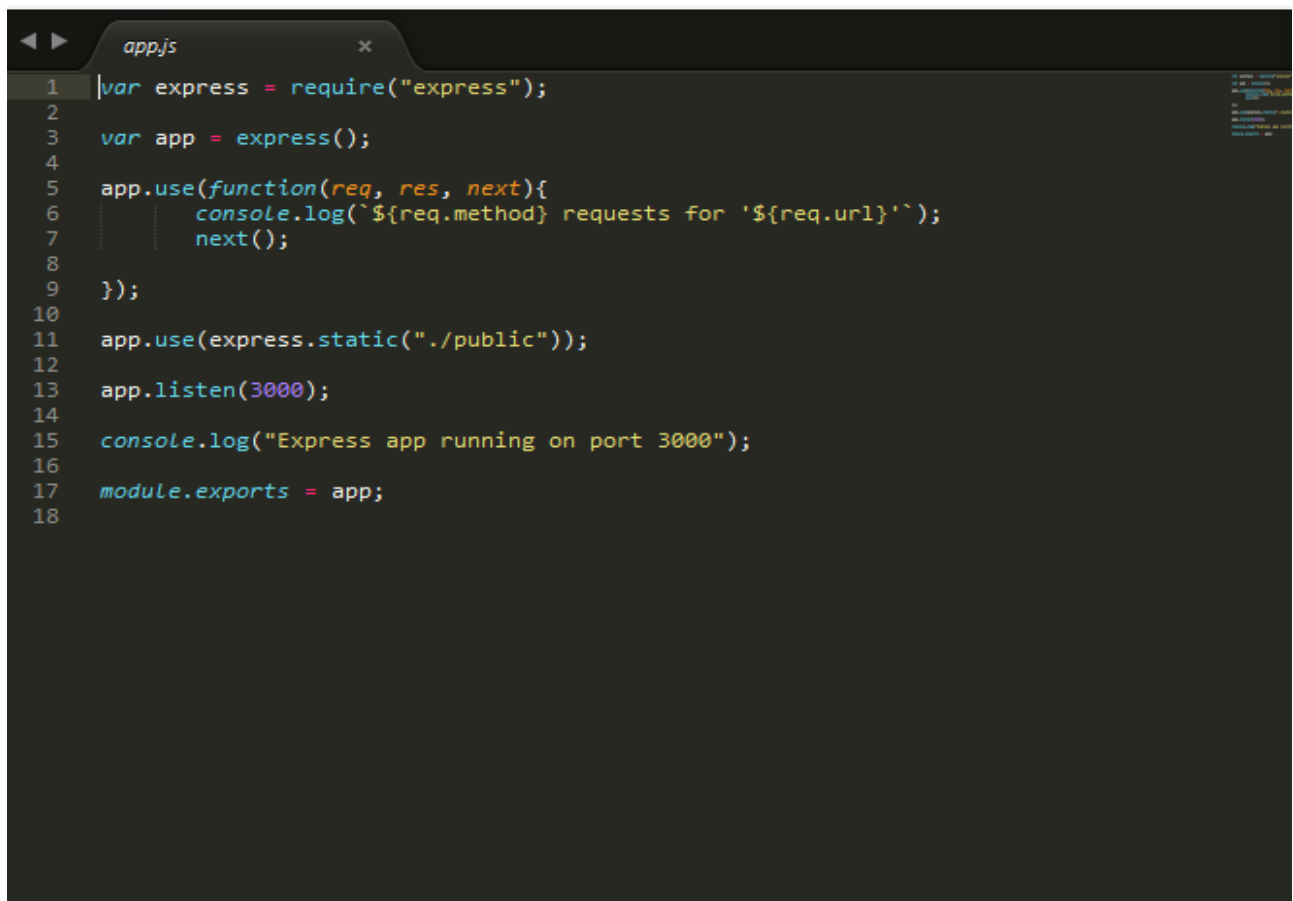
This will create a package.json file in your folder.

Now install express in myApp directory and save it in dependencies as follow

“npm install express --save”

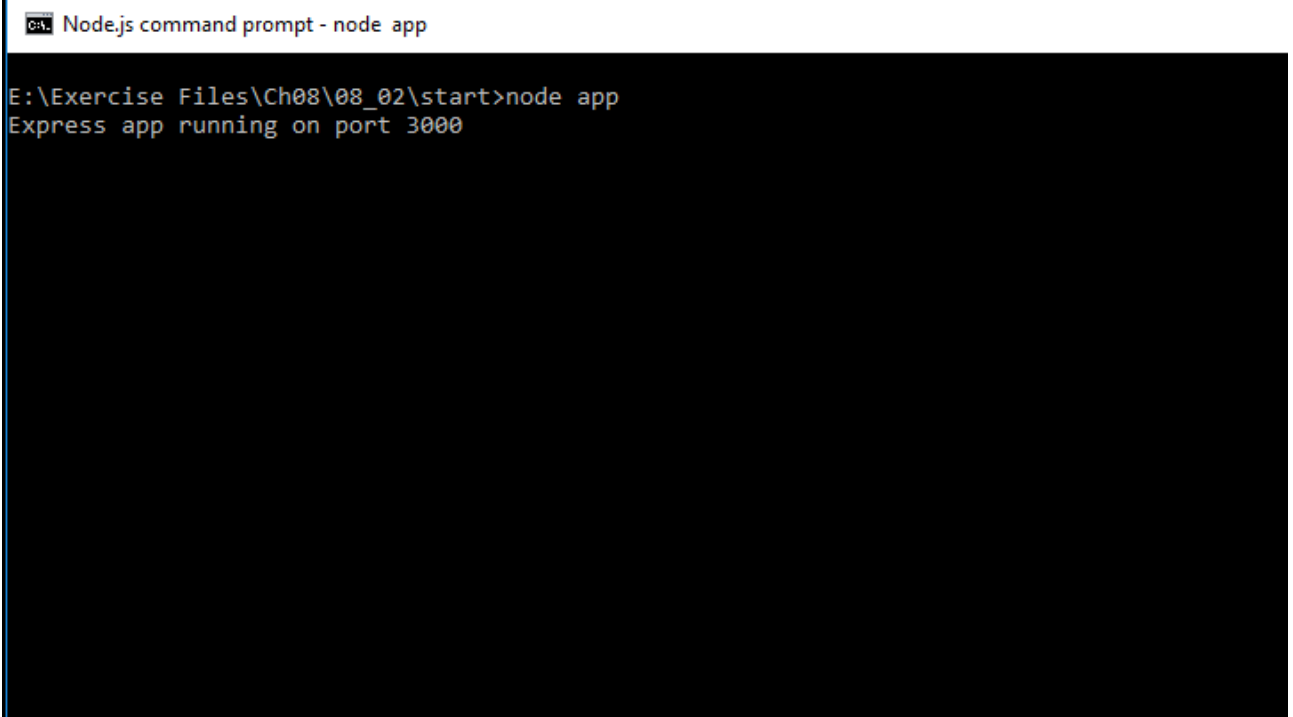
```
E:\start\myApp>npm install express --save
myapp@1.0.0 E:\start\myApp
`-- express@4.14.0
   |-- accepts@1.3.3
   |   |-- mime-types@2.1.13
   |   |   |-- mime-db@1.25.0
   |   |   |-- negotiator@0.6.1
   |-- array-flatten@1.1.1
   |-- content-disposition@0.5.1
   |-- content-type@1.0.2
   |-- cookie@0.3.1
   |-- cookie-signature@1.0.6
   |-- debug@2.2.0
   |   |-- ms@0.7.1
   |-- depd@1.1.0
   |-- encodeurl@1.0.1
   |-- escape-html@1.0.3
   |-- etag@1.7.0
   |-- finalhandler@0.5.0
   |   |-- statuses@1.3.1
   |   |-- unpipe@1.0.0
   |-- fresh@0.3.0
   |-- merge-descriptors@1.0.1
   |-- methods@1.1.2
   |-- on-finished@2.3.0
   |   |-- ee-first@1.1.1
   |-- parseurl@1.3.1
   |-- path-to-regexp@0.1.7
   |-- proxy-addr@1.1.2
   |   |-- forwarded@0.1.0
   |   |-- ipaddr.js@1.1.1
   |-- qs@6.2.0
   |-- range-parser@1.2.0
   |-- send@0.14.1
   |   |-- destroy@1.0.4
   |   |-- http-errors@1.5.1
   |   |   |-- inherits@2.0.3
   |   |   |-- setprototypeof@1.0.2
   |   |   |-- mime@1.3.4
   |-- serve-static@1.11.1
   |-- type-is@1.6.14
   |   |-- media-typer@0.3.0
   |-- utils-merge@1.0.0
```

Next create a file called app.js in start folder then open your file in that text editor and write code shown bellow

A screenshot of a code editor with a dark theme. The editor has a tab labeled 'app.js' with a close button. The code is as follows:

```
1 |var express = require("express");
2
3 |var app = express();
4
5 |app.use(function(req, res, next){
6 |    |    console.log(`${req.method} requests for '${req.url}'`);
7 |    |    next();
8 |}
9 |});
10
11 |app.use(express.static("./public"));
12
13 |app.listen(3000);
14
15 |console.log("Express app running on port 3000");
16
17 |module.exports = app;
18
```

Server will run on port 3000 as shown

A screenshot of a Node.js command prompt window. The title bar reads 'Node.js command prompt - node app'. The command prompt shows the following text:

```
E:\Exercise Files\Ch08\08_02\start>node app
Express app running on port 3000
```


Then open <http://localhost3000> in your browser



Skier Dictionary

Dictionary Empty

Testing and Debugging :-

Unit testing is used to find out hidden bugs in our application. There are several tools available for JavaScript testing like mocha, jasmine etc.

Here we are using mocha for unit testing.

Create a folder name start and install mocha as globally as following “**npm install -g mocha**”

```
Node.js command prompt

E:\start>npm install -g mocha
C:\Users\Tek2\AppData\Roaming\npm\_mocha -> C:\Users\Tek2\AppData\Roaming\npm\node_modules\mocha\bin\_mocha
C:\Users\Tek2\AppData\Roaming\npm\mocha -> C:\Users\Tek2\AppData\Roaming\npm\node_modules\mocha\bin\mocha
C:\Users\Tek2\AppData\Roaming\npm
`-- mocha@3.2.0

E:\start>
```

To run a test just go to terminal and type “mocha”

For testing any app first of all make directory named test “**mkdir test**” and go to start folder and install “**chai**” module as follow

“**npm install chai --save**”

```
Node.js command prompt

E:\start>npm install chai --save
npm WARN saveError ENOENT: no such file or directory, open 'E:\start\package.json'
E:\start
`-- chai@3.5.0
   +-- assertion-error@1.0.2
   +-- deep-eql@0.1.3
   | `-- type-detect@0.1.1
   `-- type-detect@1.0.0

npm WARN enoent ENOENT: no such file or directory, open 'E:\start\package.json'
npm WARN start No description
npm WARN start No repository field.
npm WARN start No README data
npm WARN start No license field.

E:\start>
```

We install chai module to check out result.

Now create a file name tools.js in lib folder as shown



The screenshot shows a code editor with two tabs: 'tools-spec.js' and 'tools.js'. The 'tools.js' tab is active, displaying the following JavaScript code:

```
1 module.exports = {
2
3   printName(person){
4     return `${person.last}, ${person.first}`;
5   }
6 }
7 };
```

Now create a tools.specs.js file in test folder as shown below.



The screenshot shows a code editor with a single tab: 'tools-spec.js'. The file contains the following JavaScript code for testing:

```
1 var expect = require("chai").expect;
2 var tools = require("../lib/tools");
3
4
5 describe("printName()", function(){
6
7   it("should first name last", function(){
8     var results = tools.printName({first: "Ashok", last: "Vyas"});
9
10    expect(results).to.equal("Vyas, Ashok");
11  });
12 });
13 };
```

Now go to terminal window and run the test by typing “mocha”

```
Node.js command prompt
E:\Exercise Files\Ch10\10_01\start>mocha

  printName()
    ✓ should first name last

  1 passing (31ms)

E:\Exercise Files\Ch10\10_01\start>
```

As we can see test for print function passed and print function working properly.

Mocking a server with Nock:-

Some time testing takes little bit more time example like if we are testing some downloading pages etc. like activity. So to reduce the time and making testing more faster we use a concept of mocking. There a tool called nock which allows us to mocking.

First we install nock module, so create a folder named start and go to terminal and type

“npm install nock --save-dev”

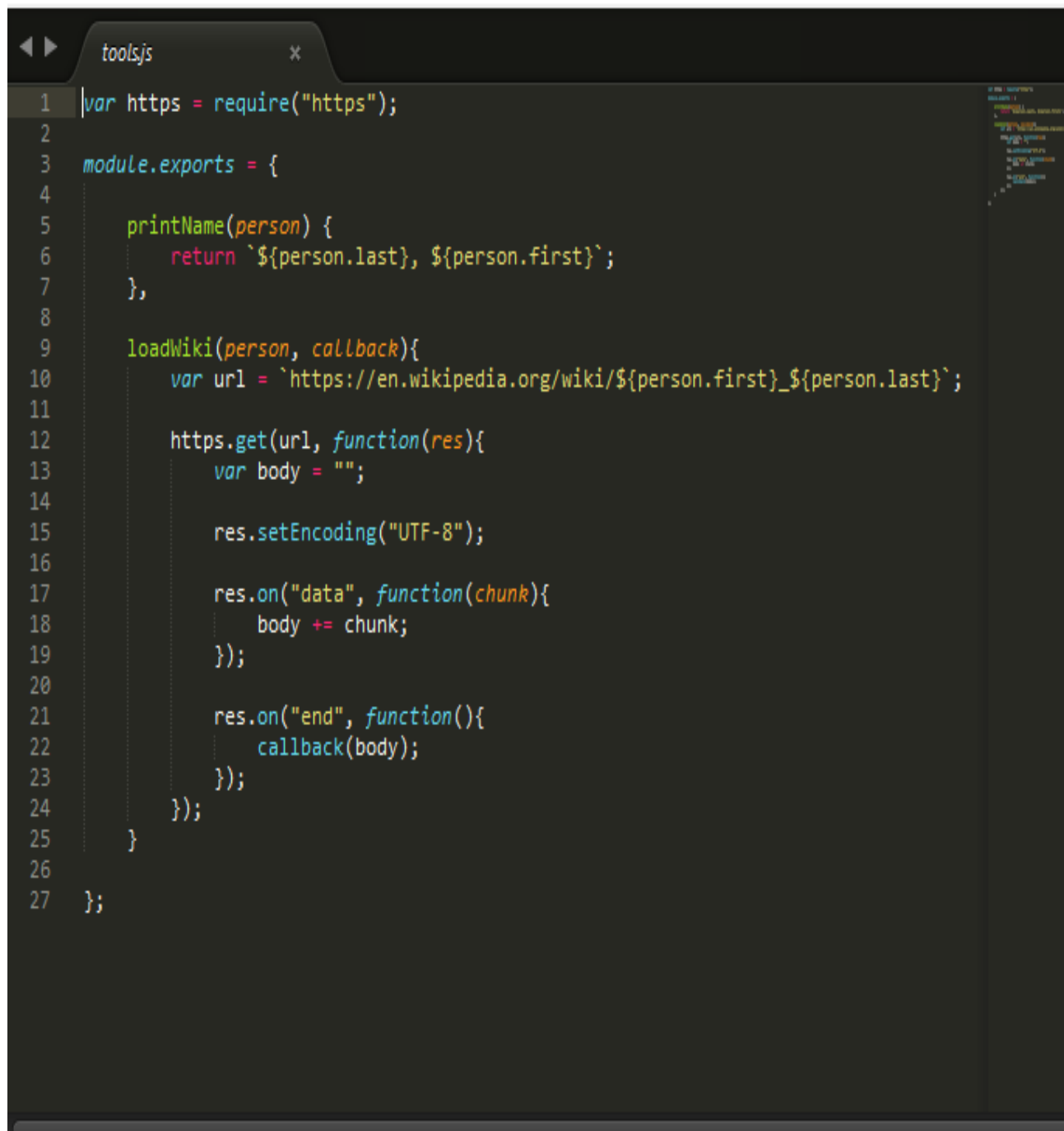
```
E:\Exercise Files\Ch10\10_01\start>npm install nock --save-dev
mocha-testing@1.0.0 E:\Exercise Files\Ch10\10_01\start
`-- nock@9.0.2
   +-- debug@2.4.4
   |   `-- ms@0.7.2
   +-- deep-equal@1.0.1
   +-- json-stringify-safe@5.0.1
   +-- lodash@4.9.0
   +-- mkdirp@0.5.1
   |   `-- minimist@0.0.8
   +-- propagate@0.4.0
   `-- qs@6.3.0

npm WARN mocha-testing@1.0.0 No repository field.

E:\Exercise Files\Ch10\10_01\start>
```

This will save nock as dependencies.

Then go to tools.js file and add loadWiki() function as shown



```
1 var https = require("https");
2
3 module.exports = {
4
5     printName(person) {
6         return `${person.last}, ${person.first}`;
7     },
8
9     loadWiki(person, callback){
10         var url = `https://en.wikipedia.org/wiki/${person.first}_${person.last}`;
11
12         https.get(url, function(res){
13             var body = "";
14
15             res.setEncoding("UTF-8");
16
17             res.on("data", function(chunk){
18                 body += chunk;
19             });
20
21             res.on("end", function(){
22                 callback(body);
23             });
24         });
25     }
26
27 };
```

Then go to tools-spec.js file and write some code as shown below

A screenshot of a code editor with a dark theme. The editor has a tab at the top labeled 'tools-spec.js' with a close button 'x'. The code is written in JavaScript and uses Jest for testing. It includes imports for 'chai' (expect), 'tools' (from '../lib/tools'), and 'nock'. The code is structured with nested 'describe' and 'it' blocks. The first 'describe' block is for 'Tools' and contains two 'it' blocks. The second 'it' block is for 'printName()' and tests that the last name is printed first. The third 'it' block is for 'loadWiki()' and tests that the correct HTML is returned. The 'loadWiki()' test uses a 'before' block to set up a mock with 'nock'.

```
1 |var expect = require("chai").expect;
2 |var tools = require("../lib/tools");
3 |var nock = require("nock");
4
5 |describe("Tools", function() {
6 |
7 |    describe("printName()", function() {
8 |        it("should print the last name first", function() {
9 |            var results = tools.printName({ first: "Alex", last: "Banks"});
10 |            expect(results).to.equal("Banks, Alex");
11 |        });
12 |    });
13
14 |    describe("loadWiki()", function() {
15 |
16 |        before(function(){
17 |            nock("https://en.wikipedia.org")
18 |                .get("/wiki/Swami_Vivekanand")
19 |                .reply(200, "Mock Swami Vivekanand page");
20 |        });
21
22 |        it("Load Swami Vivekanand's wikipedia page", function(done) {
23 |
24 |            tools.loadWiki({ first: "Swami", last: "Vivekanand"}, function(html) {
25 |                expect(html).to.equal("Mock Swami Vivekanand page");
26 |                done();
27 |            });
28 |
29 |        });
30 |
31 |    });
32
33 |});
```

Next go to terminal and run the test and we see both test printName and loadwiki both passed successfully.

Node.js command prompt

```
E:\Exercise Files\Ch10\10_03\start>mocha

Tools
  printName()
    ✓ should print the last name first
  loadWiki()
    ✓ Load Swami Vivekanand's wikipedia page

2 passing (15ms)

E:\Exercise Files\Ch10\10_03\start>
```

Rewire module:-

Rewire is a node module which allows us to inject our mocks. In this, write a test and we use our mock data and inject that data to real data and use that mock data for testing.

First of all install rewire module and save as dev dependencies so create a folder name start and go to terminal

“npm install rewire --save-dev”

Node.js command prompt

```
E:\start>npm install rewire --save-dev
npm WARN saveError ENOENT: no such file or directory, open 'E:\start\package.json'
E:\start
└-- rewire@2.5.2

npm WARN enoent ENOENT: no such file or directory, open 'E:\start\package.json'
npm WARN start No description
npm WARN start No repository field.
npm WARN start No README data
npm WARN start No license field.

E:\start>
```


Sinon Spies:-

Sinon provides spies, stubs, and mocks. They're all useful as fakes in tests. They come with essential differences for what they're helpful in doing and how they work.

Spies sound like what they do, they watch your functions and report back on how they are called. They don't change the functionality of your application. They simply report what they see. This is fairly large, but it essentially centres around the called attribute (of which there are many variations).

To install sinon create a folder and go to terminal and type

“npm install sinon --save-dev”

 Node.js command prompt

```
E:\start>npm install sinon --save-dev
npm WARN saveError ENOENT: no such file or directory, open 'E:\start\package.json'
E:\start
`-- sinon@1.17.6
   +-- formatio@1.1.1
   +-- lolex@1.3.2
   +-- samsam@1.1.2
   `-- util@0.10.3
      `-- inherits@2.0.1

npm WARN enoent ENOENT: no such file or directory, open 'E:\start\package.json'
npm WARN start No description
npm WARN start No repository field.
npm WARN start No README data
npm WARN start No license field.

E:\start>
```


Sinon stub:-

Stubs are more hands on than spies though they sound more useless. With a stub, you will actually change how functions are called in your test. You don't want to change the subject under test, thus changing the accuracy of your test. But you may want to test several ways that dependencies of your unit could be expected to act.

Using sinon stub we are going to test a order function present in our order module as shown below

```
order.js
1 | var inventoryData = require('../data/inventory');
2 | var warehouse = require('../warehouse');
3 |
4 | function findItem(sku) {
5 |     var i = inventoryData.map(item => item.sku).indexOf(sku);
6 |     if (i === -1) {
7 |         console.log(`Item - ${sku} not found`);
8 |         return null;
9 |     } else {
10 |         return inventoryData[i];
11 |     }
12 | }
13 |
14 | function isInStock(sku, qty) {
15 |     var item = findItem(sku);
16 |     return item && item.qty >= qty;
17 | }
18 |
19 | function order(sku, quantity, complete) {
20 |     complete = complete || function () {};
21 |     if (isInStock(sku, quantity)) {
22 |         console.log(`ordering ${quantity} of item # ${sku}`);
23 |         warehouse.packageAndShip(sku, quantity, function (tracking) {
24 |             console.log(`order shipped, tracking - ${tracking}`);
25 |             complete(tracking);
26 |         });
27 |         return true;
28 |     } else {
29 |         console.log(`there are not ${quantity} of item '${sku}' in stock`);
30 |         return false;
31 |     }
32 | }
33 |
34 | module.exports.orderItem = order;
```

Next create a order-spec.js file as shown below and write a test

```
order-spec.js x
1 |var expect = require("chai").expect;
2 |var rewire = require("rewire");
3
4 |var order = rewire("../lib/order");
5
6 |var sinon = require("sinon");
7
8 |describe("Ordering Items", function() {
9
10 |    beforeEach(function() {
11
12 |        this.testData = [
13 |            {sku: "AAA", qty: 10},
14 |            {sku: "BBB", qty: 0},
15 |            {sku: "CCC", qty: 3}
16 |        ];
17
18 |        this.console = {
19 |            log: sinon.spy()
20 |        };
21
22 |        this.warehouse = {
23 |            packageAndShip: sinon.stub().yields(10987654321)
24 |        };
25
26 |        order.__set__("inventoryData", this.testData);
27 |        order.__set__("console", this.console);
28 |        order.__set__("warehouse", this.warehouse);
29
30 |    });
31
32 |    it("order an item when there are enough in stock", function(done) {
33
34 |        var _this = this;
35
36 |        order.orderItem("CCC", 3, function() {
37
38 |            expect(_this.console.log.callCount).to.equal(2);
39
40 |            done();
41 |        });
42
43 |    });
44
```

Supertest module:-

Supertest module is used to test the http application such as express site.

So create a folder called start and install supertest module as follow

“npm install supertest –save-dev”

Then create a app.js file and app-spec.js file inside your start folder

```
app.js
1 | var express = require("express");
2 | var cors = require("cors");
3 | var bodyParser = require("body-parser");
4 | var app = express();
5 |
6 | var skierTerms = [
7 |   {
8 |     term: "Rip",
9 |     defined: "To move at a high rate of speed"
10 |   },
11 |   {
12 |     term: "Huck",
13 |     defined: "To throw your body off of something, usually a natural feature like a cliff"
14 |   },
15 |   {
16 |     term: "Chowder",
17 |     defined: "Powder after it has been sufficiently skied"
18 |   }
19 | ];
20 |
21 | app.use(bodyParser.json());
22 | app.use(bodyParser.urlencoded({ extended: false }));
23 |
24 | app.use(function(req, res, next) {
25 |   console.log(`${req.method} request for '${req.url}' - ${JSON.stringify(req.body)}`);
26 |   next();
27 | });
28 |
29 | app.use(express.static("./public"));
30 |
31 | app.use(cors());
32 |
33 | app.get("/dictionary-api", function(req, res) {
34 |   res.json(skierTerms);
35 | });
36 |
37 | app.post("/dictionary-api", function(req, res) {
38 |   skierTerms.push(req.body);
39 |   res.json(skierTerms);
40 | });
41 |
42 | app.delete("/dictionary-api/:term", function(req, res) {
43 |   skierTerms = skierTerms.filter(function(definition) {
44 |     return definition.term.toLowerCase() !== req.params.term.toLowerCase();
45 |   });
```

```
46 |     res.json(skierTerms);
47 |   });
48 |
49 | app.listen(3000);
50 |
51 | console.log("Express app running on port 3000");
52 |
53 | module.exports = app;
```

```
app-spec.js x
1 | var expect = require('chai').expect;
2 | var request = require("supertest");
3 | var rewire = require('rewire');
4 | var app = rewire('../app');
5
6 ▼ describe("Dictionary App", function () {
7
8     it("Loads the home page", function(done){
9         request(app).get("/").expect(200).end(done);
10    });
11
12 ▼ describe("Dictionary API", function () {
13
14 ▼     beforeEach(function () {
15
16 ▼         this.defs = [
17 ▼             {
18                 term: "One",
19                 defined: "Term One Defined"
20             },
21 ▼             {
22                 term: "Two",
23                 defined: "Term Two Defined"
24             }
25         ];
26
27         app.__set__("skierTerms", this.defs);
28     });
29
30     it("GETS dictionary-api", function(done){
31         request(app).get("/dictionary-api").expect(200).end(done);
32     });
33
34 ▼ it("POSTS dictionary-api", function(done){
35 ▼     request(app)
36         .post("/dictionary-api")
37         .send({"term": "Three", "defined": "Three defined"})
38         .expect(200)
39         .end(done);
40     });
41
```

```
41
42     it("DELETES dictionary-api", function(done){
43         request(app)
44             .delete("/dictionary-api/One")
45             .expect(200)
46             .end(done);
47     });
48
49 });
50
51 });
```

Then go to terminal and test the app using mocha

```
Node.js command prompt

E:\Exercise Files\Ch10\10_08\start>mocha
Express app running on port 3000

  Dictionary App
    GET request for '/' - {}
      ✓ Loads the home page
    Dictionary API
      GET request for '/dictionary-api' - {}
        ✓ GETS dictionary-api
      POST request for '/dictionary-api' - {"term":"Three","defined":"Three defined"}
        ✓ POSTS dictionary-api
      DELETE request for '/dictionary-api/One' - {}
        ✓ DELETES dictionary-api

  4 passing (84ms)

E:\Exercise Files\Ch10\10_08\start>
```

As we see all the test are passed.

Automation and Deployment:-

Hinting your code with grunt:-

Create a folder name start and install grunt globally as shown
“npm install -g grunt-cli”

```
Node.js command prompt

E:\start>npm install -g grunt-cli
C:\Users\Tek2\AppData\Roaming\npm\grunt -> C:\Users\Tek2\AppData\Roaming\npm\node_modules\grunt-cli\bin\grunt
C:\Users\Tek2\AppData\Roaming\npm
^-- grunt-cli@1.2.0

E:\start>
```

Next we have to install grunt locally where we want to use so create a file named app.js and go to terminal and install grunt locally as shown

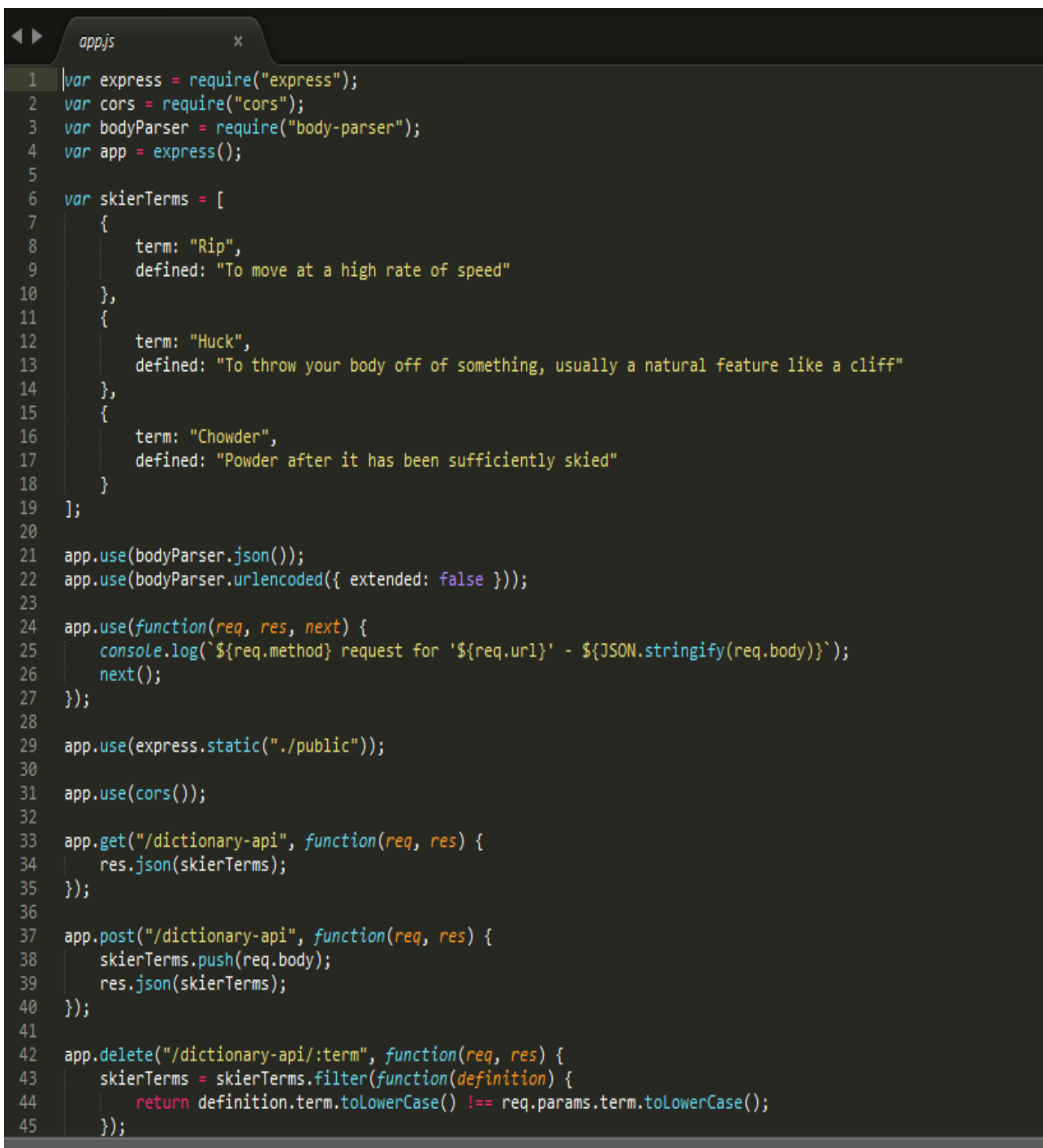
“npm install grunt --save-dev”

```
E:\start>npm install grunt --save-dev
npm WARN saveError ENOENT: no such file or directory, open 'E:\start\package.json'
E:\start
`-- grunt@1.0.1
   +-- coffee-script@1.10.0
   +-- dateformat@1.0.12
   | +-- get-stdin@4.0.1
   | `-- meow@3.7.0
   |   +-- camelcase-keys@2.1.0
   |   | `-- camelcase@2.1.1
   |   +-- decamelize@1.2.0
   |   +-- loud-rejection@1.6.0
   |   | +-- currently-unhandled@0.4.1
   |   | | `-- array-find-index@1.0.2
   |   | `-- signal-exit@3.0.2
   |   +-- map-obj@1.0.1
   |   +-- minimist@1.2.0
   |   +-- normalize-package-data@2.3.5
   |   | +-- hosted-git-info@2.1.5
   |   | +-- is-builtin-module@1.0.0
   |   | | `-- builtin-modules@1.1.1
   |   | +-- semver@5.3.0
   |   | `-- validate-npm-package-license@3.0.1
   |   |   +-- spdx-correct@1.0.2
   |   |   | `-- spdx-license-ids@1.2.2
   |   |   `-- spdx-expression-parse@1.0.4
   |   +-- object-assign@4.1.0
   |   +-- read-pkg-up@1.0.1
   |   | +-- find-up@1.1.2
```


For hinting the code we also need to install jshint

“npm install grunt-contrib-jshint --save-dev”

Next create a folder named start and include a file named app.js and write code as shown below



```
1 var express = require("express");
2 var cors = require("cors");
3 var bodyParser = require("body-parser");
4 var app = express();
5
6 var skierTerms = [
7   {
8     term: "Rip",
9     defined: "To move at a high rate of speed"
10  },
11  {
12    term: "Huck",
13    defined: "To throw your body off of something, usually a natural feature like a cliff"
14  },
15  {
16    term: "Chowder",
17    defined: "Powder after it has been sufficiently skied"
18  }
19 ];
20
21 app.use(bodyParser.json());
22 app.use(bodyParser.urlencoded({ extended: false }));
23
24 app.use(function(req, res, next) {
25   console.log(`${req.method} request for '${req.url}' - ${JSON.stringify(req.body)}`);
26   next();
27 });
28
29 app.use(express.static("./public"));
30
31 app.use(cors());
32
33 app.get("/dictionary-api", function(req, res) {
34   res.json(skierTerms);
35 });
36
37 app.post("/dictionary-api", function(req, res) {
38   skierTerms.push(req.body);
39   res.json(skierTerms);
40 });
41
42 app.delete("/dictionary-api/:term", function(req, res) {
43   skierTerms = skierTerms.filter(function(definition) {
44     return definition.term.toLowerCase() !== req.params.term.toLowerCase();
45   });
46 });
```

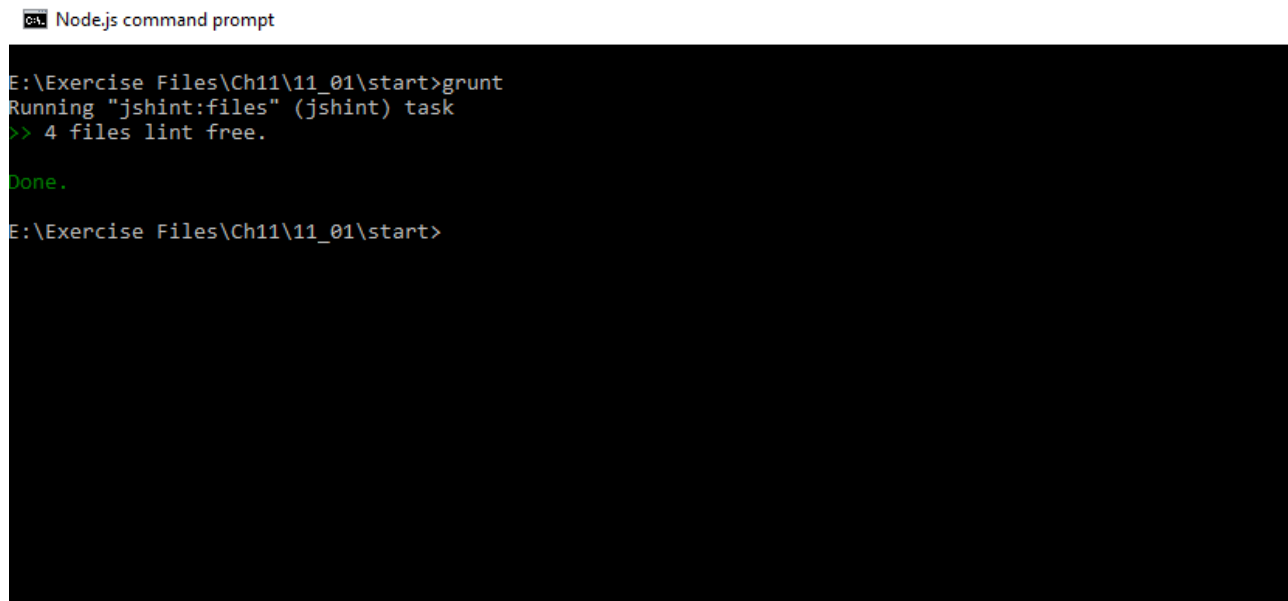
```
46 |     res.json(skierTerms);
47 | });
48 |
49 | app.listen(3000);
50 |
51 | console.log("Express app running on port 3000");
52 |
53 | module.exports = app;
```

And a GruntFile.js as shown

A screenshot of a code editor window titled 'GruntFile.js'. The code is written in JavaScript and configures the Grunt task runner. It includes a 'jshint' task configuration with file patterns and options like 'esnext' and 'globals'. The code is as follows:

```
1 module.exports = function(grunt){
2
3   grunt.initConfig({
4     jshint: {
5       files: ["*.js", "lib/*.js", "text/*.js"],
6       options:{
7         esnext: true,
8         globals: true
9       }
10    }
11  });
12
13  grunt.loadNpmTasks("grunt-contrib-jshint");
14
15  grunt.registerTask("default", ["jshint"]);
16 };
```

Then go to terminal and type grunt

A screenshot of a Node.js command prompt terminal. The terminal shows the execution of the 'grunt' command in the directory 'E:\Exercise Files\Ch11\11_01\start'. The output indicates that the 'jshint:files' task was run successfully, linting 4 files without any errors. The terminal text is as follows:

```
C:\> Node.js command prompt

E:\Exercise Files\Ch11\11_01\start>grunt
Running "jshint:files" (jshint) task
>> 4 files lint free.

Done.

E:\Exercise Files\Ch11\11_01\start>
```

Here we done jshint on four files and files are OK.

Converting less to css:-

There is a plug-in available to convert less file to css. So create a folder and install a grunt-contrib-less as shown below

“npm install grunt-config-less --save-dev ”

and save that to dev- dependencies. then use the below code to convert any less file to css file.

GruntFile.js

```
1 module.exports = function(grunt) {
2
3   grunt.initConfig({
4     jshint: {
5       files: ["*.js", "lib/*.js", "test/*.js"],
6       options: {
7         esnext: true,
8         globals: {
9           jQuery: true
10        }
11      }
12    },
13
14    less: {
15      production: {
16        files: {
17          "public/css/style.css": ["less/*.less"]
18        }
19      }
20    },
21
22    autoprefixer: {
23      single_file: {
24        src: "public/css/style.css",
25        dest: "public/css/style.css"
26      }
27    }
28  });
29
30  grunt.loadNpmTasks("grunt-contrib-jshint");
31  grunt.loadNpmTasks("grunt-contrib-less");
32  grunt.loadNpmTasks("grunt-autoprefixer");
33
34  grunt.registerTask("css", ["less", "autoprefixer"]);
35
36  grunt.registerTask("default", ["jshint", "css"]);
37  };
```

Rapid development with grunt watches:-

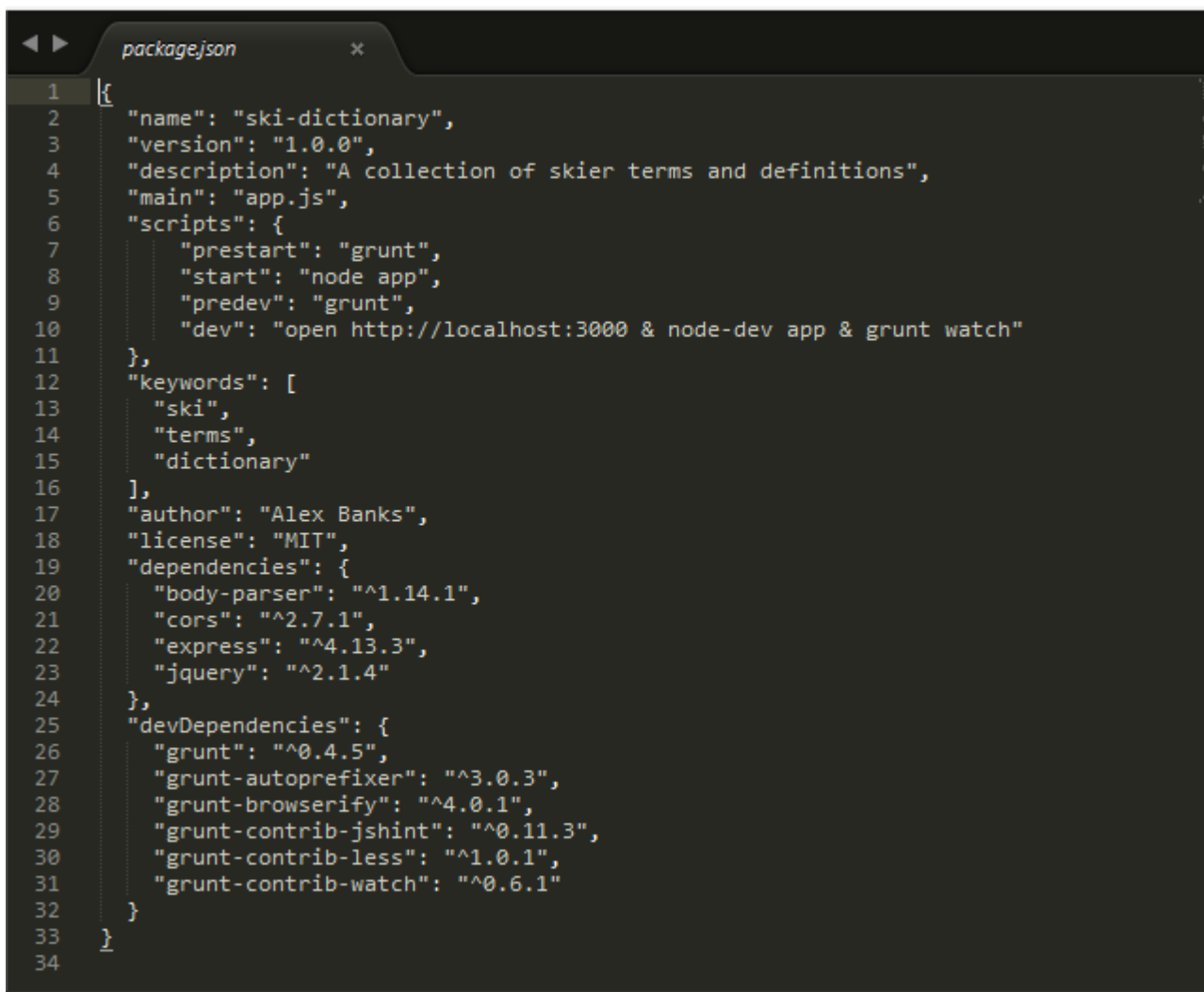
We can use grunt for watching changes in files during development. So we have to install grunt-contrib-watches to have this functionality.

```
“npm install grunt-contrib-watch –save-dev”
```

If we do any changes in less file or any other files it automatically watch that changes and fix them..

Automation with npm:-

NPM also provide automatic running, debugging, testing etc.



```
1 {
2   "name": "ski-dictionary",
3   "version": "1.0.0",
4   "description": "A collection of skier terms and definitions",
5   "main": "app.js",
6   "scripts": {
7     "prestart": "grunt",
8     "start": "node app",
9     "predev": "grunt",
10    "dev": "open http://localhost:3000 & node-dev app & grunt watch"
11  },
12  "keywords": [
13    "ski",
14    "terms",
15    "dictionary"
16  ],
17  "author": "Alex Banks",
18  "license": "MIT",
19  "dependencies": {
20    "body-parser": "^1.14.1",
21    "cors": "^2.7.1",
22    "express": "^4.13.3",
23    "jquery": "^2.1.4"
24  },
25  "devDependencies": {
26    "grunt": "^0.4.5",
27    "grunt-autoprefixer": "^3.0.3",
28    "grunt-browserify": "^4.0.1",
29    "grunt-contrib-jshint": "^0.11.3",
30    "grunt-contrib-less": "^1.0.1",
31    "grunt-contrib-watch": "^0.6.1"
32  }
33 }
34
```

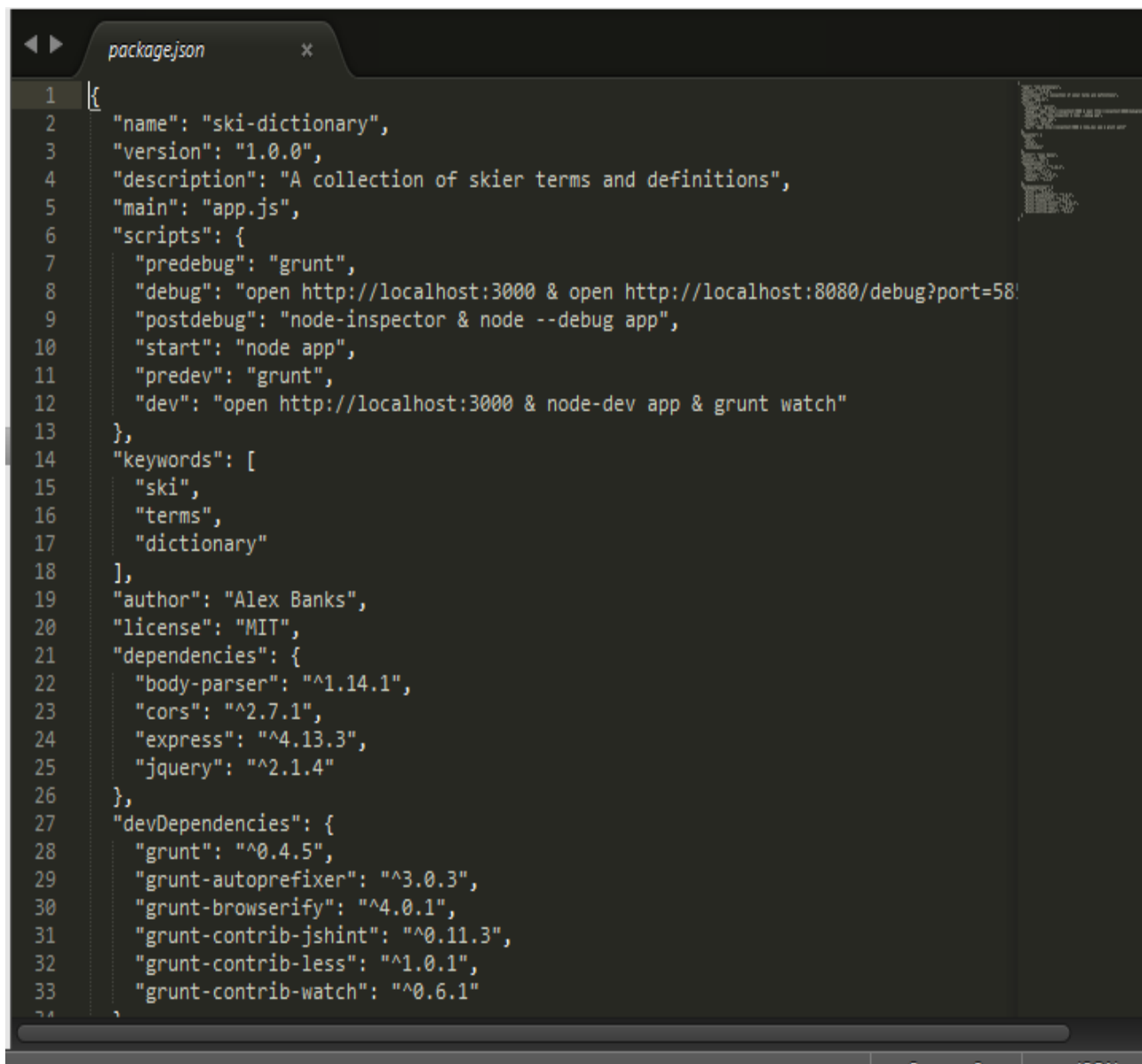
As we see in package json file we can run app using” npm start” instead of node app and there is pre start function so before running app grunt will run.

Debugging with npm:-

There is a tool available to debugging in Nodejs called node-inspector. To install node-inspector go to terminal and install it globally as follow

“npm install -g node-inspector”

and we also have to add debug in package json file and give a port where to listen the debugging.

A screenshot of a code editor window titled 'package.json'. The editor shows a JSON configuration for a project named 'ski-dictionary'. The 'scripts' section is configured for debugging with node-inspector. The 'dev' script runs 'open http://localhost:3000 & node-dev app & grunt watch'. The 'debug' script runs 'open http://localhost:3000 & open http://localhost:8080/debug?port=5858'. The 'postdebug' script runs 'node-inspector & node --debug app'. The 'start' script runs 'node app'. The 'predev' script runs 'grunt'. The 'predebug' script runs 'grunt'. The 'dependencies' section lists 'body-parser', 'cors', 'express', and 'jquery'. The 'devDependencies' section lists 'grunt', 'grunt-autoprefixer', 'grunt-browserify', 'grunt-contrib-jshint', 'grunt-contrib-less', and 'grunt-contrib-watch'.

```
1 {
2   "name": "ski-dictionary",
3   "version": "1.0.0",
4   "description": "A collection of skier terms and definitions",
5   "main": "app.js",
6   "scripts": {
7     "predebug": "grunt",
8     "debug": "open http://localhost:3000 & open http://localhost:8080/debug?port=5858",
9     "postdebug": "node-inspector & node --debug app",
10    "start": "node app",
11    "predev": "grunt",
12    "dev": "open http://localhost:3000 & node-dev app & grunt watch"
13  },
14  "keywords": [
15    "ski",
16    "terms",
17    "dictionary"
18  ],
19  "author": "Alex Banks",
20  "license": "MIT",
21  "dependencies": {
22    "body-parser": "^1.14.1",
23    "cors": "^2.7.1",
24    "express": "^4.13.3",
25    "jquery": "^2.1.4"
26  },
27  "devDependencies": {
28    "grunt": "^0.4.5",
29    "grunt-autoprefixer": "^3.0.3",
30    "grunt-browserify": "^4.0.1",
31    "grunt-contrib-jshint": "^0.11.3",
32    "grunt-contrib-less": "^1.0.1",
33    "grunt-contrib-watch": "^0.6.1"
34  },
35  }
```