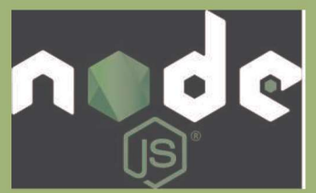
Node JS



Node.js Introduction

What Is Node JS ?

Node JS:

- Is an open-source server side runtime environment built on Chrome's V8 JavaScript Engine
- Uses Event Driven, non-blocking model that makes it lightweight and efficient.
- Node JS package Eco System, NPM, is the largest Eco System of Open Source libraries in the World.
- Cross-platform runtime environment for building highly scalable server-side application using JavaScript.

Cont …

- Node.js can be used to build different types of applications such as command line application, web application, real-time chat application, REST API server etc.

- However, it is mainly used to build network programs like web servers, similar to PHP, Java, or ASP.NET.

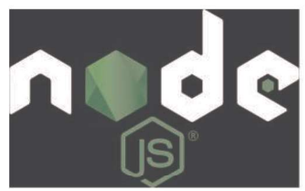
- Node.js was written and introduced by Ryan Dahl in 2009.

Advantages of Node.js

1. Node.js is an open-source framework under MIT license. (MIT license is a free software license originating at the Massachusetts Institute of Technology (MIT).)
2. Uses JavaScript to build entire server side application.
3. Lightweight framework that includes bare minimum modules. Other modules can be included as per the need of an application.
4. Asynchronous by default. So it performs faster than other frameworks.
5. Cross-platform framework that runs on Windows, MAC or Linux

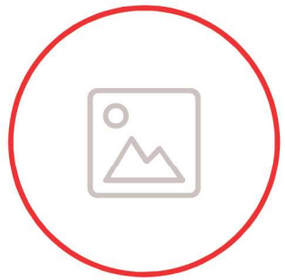
Node JS Actually is

- A Platform which allows us to run JavaScript on a Computer / Server
- Read, delete and Update Files
- Easily Communicates With a Database

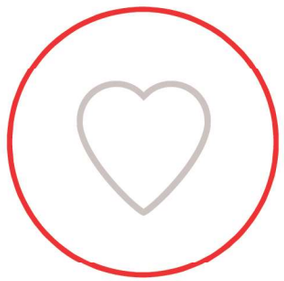


Node Secret Sauce - Code once, deploy anywhere

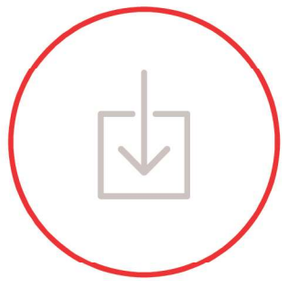
Desktop



IOT Smart Devices



Browser

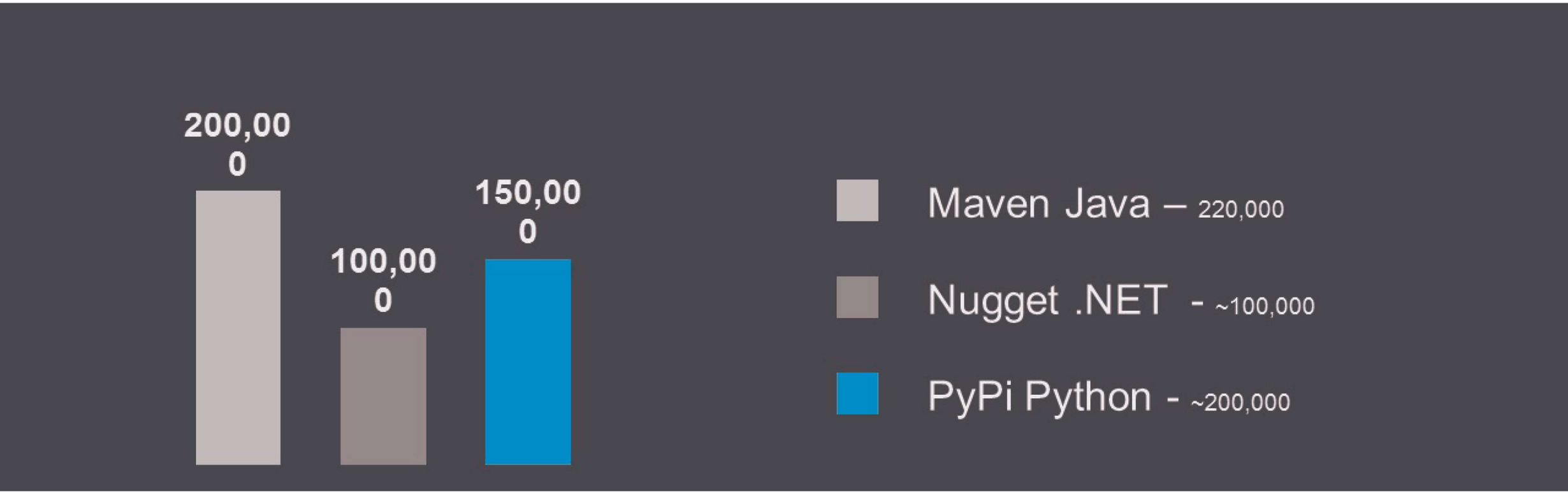


Web Server

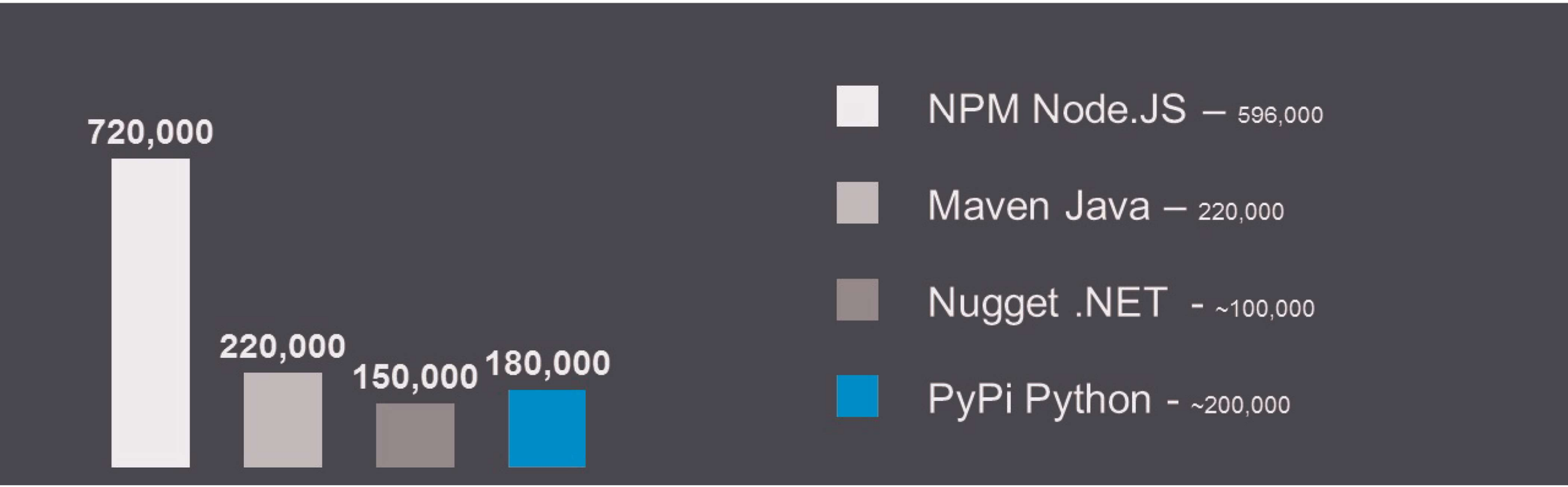
Why is Node JS so Popular ?

- It Uses JavaScript
- Very Fast (runs on the V8 Engine & uses non-blocking Code)
- Huge Eco System of Open Source Packages (npm)
- Great for Real-time Services (Like Chat Based Applications)

Platforms Packages Compared



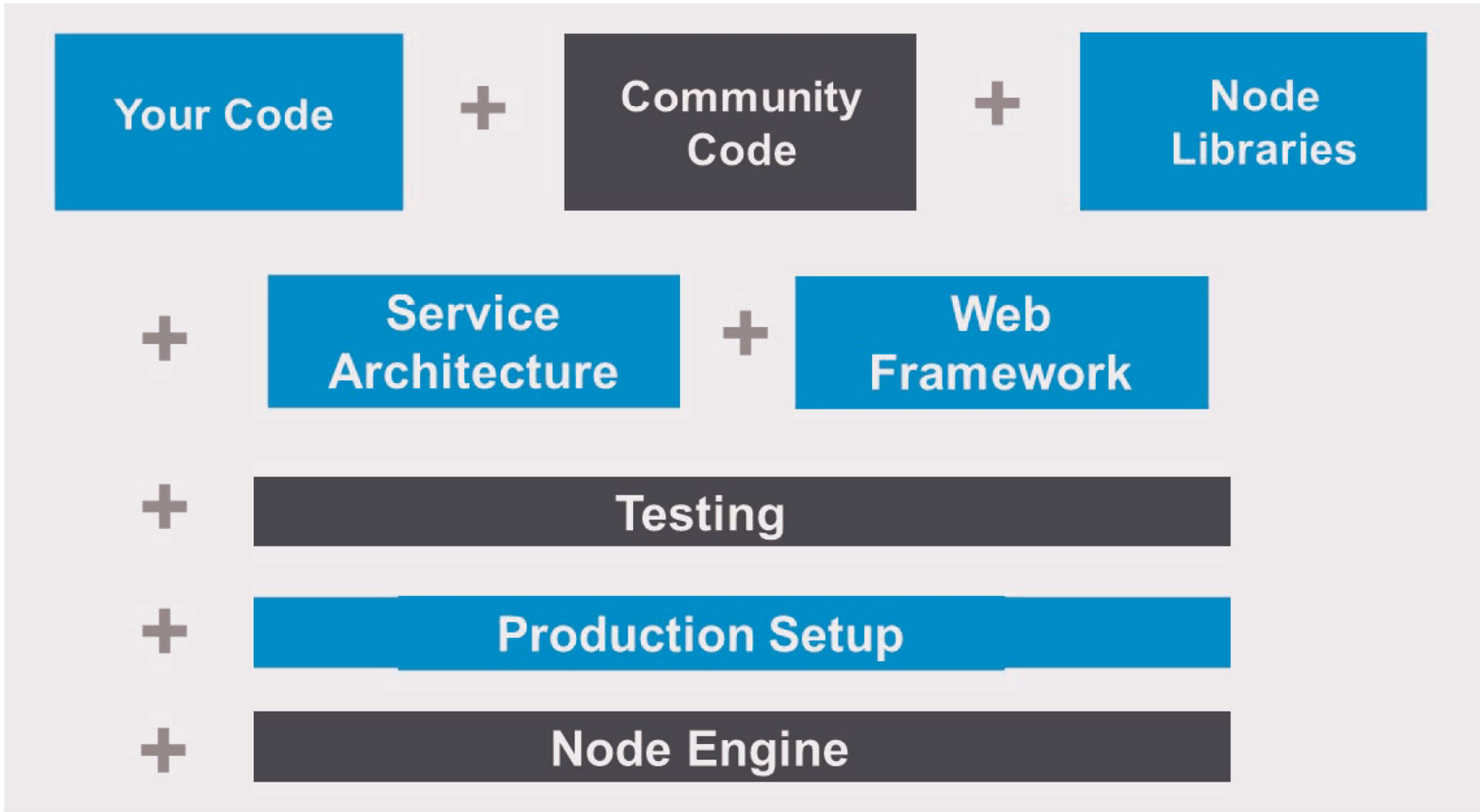
Platforms Packages Compared



Prerequisites

- JavaScript
- HTML
- A Tiny Bit about the command Line

The World Of Node.JS



How to Download & Install Node.js

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

Cont …

- Node.js also has the ability to embedded external functionality or extended functionality by making use of custom modules.

- These modules have to be installed separately. An example of a module is the MongoDB module which allows you to work with MongoDB databases from your Node.js application.

Installation

Step 1 :

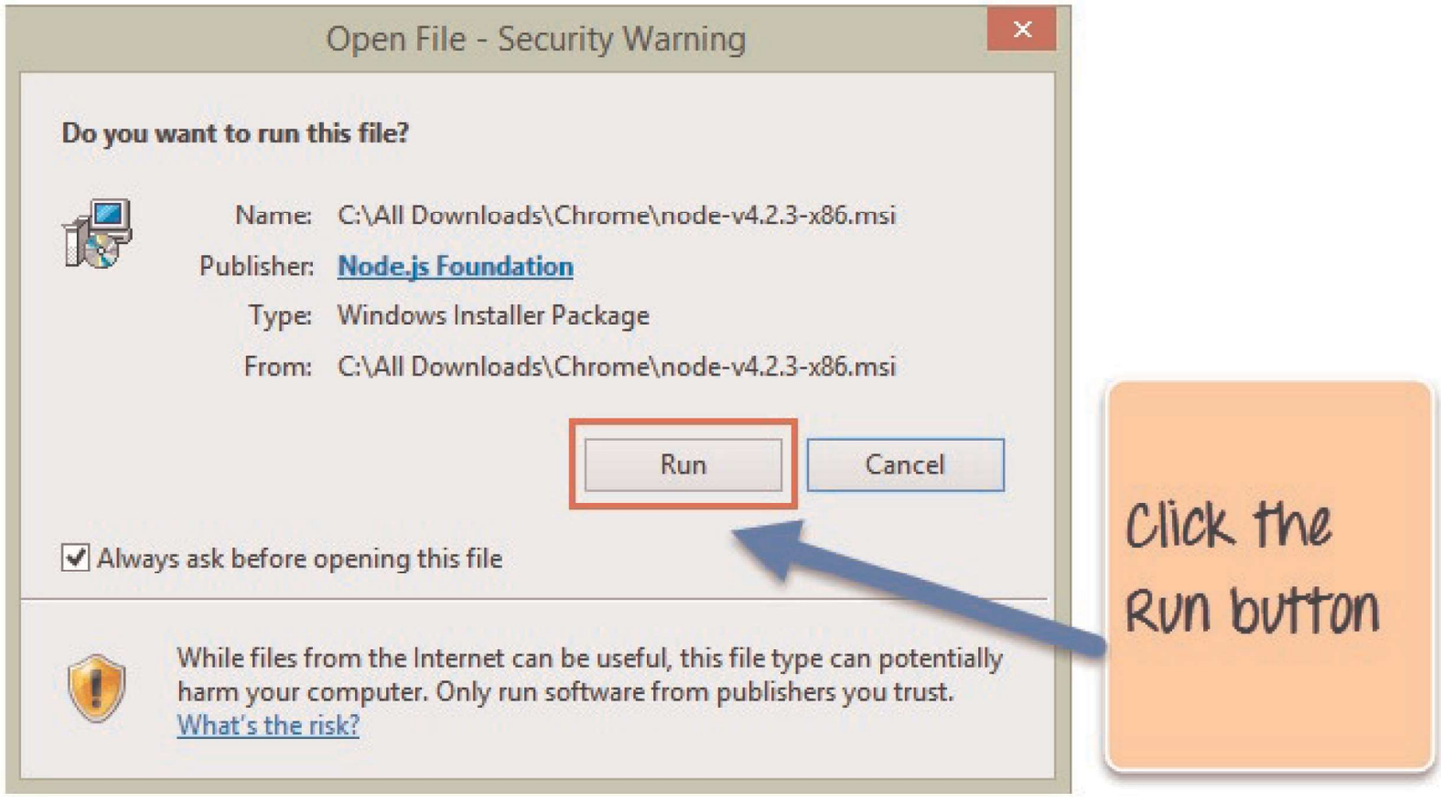
Downloads

Latest LTS Version: **10.16.3** (includes npm 6.9.0)

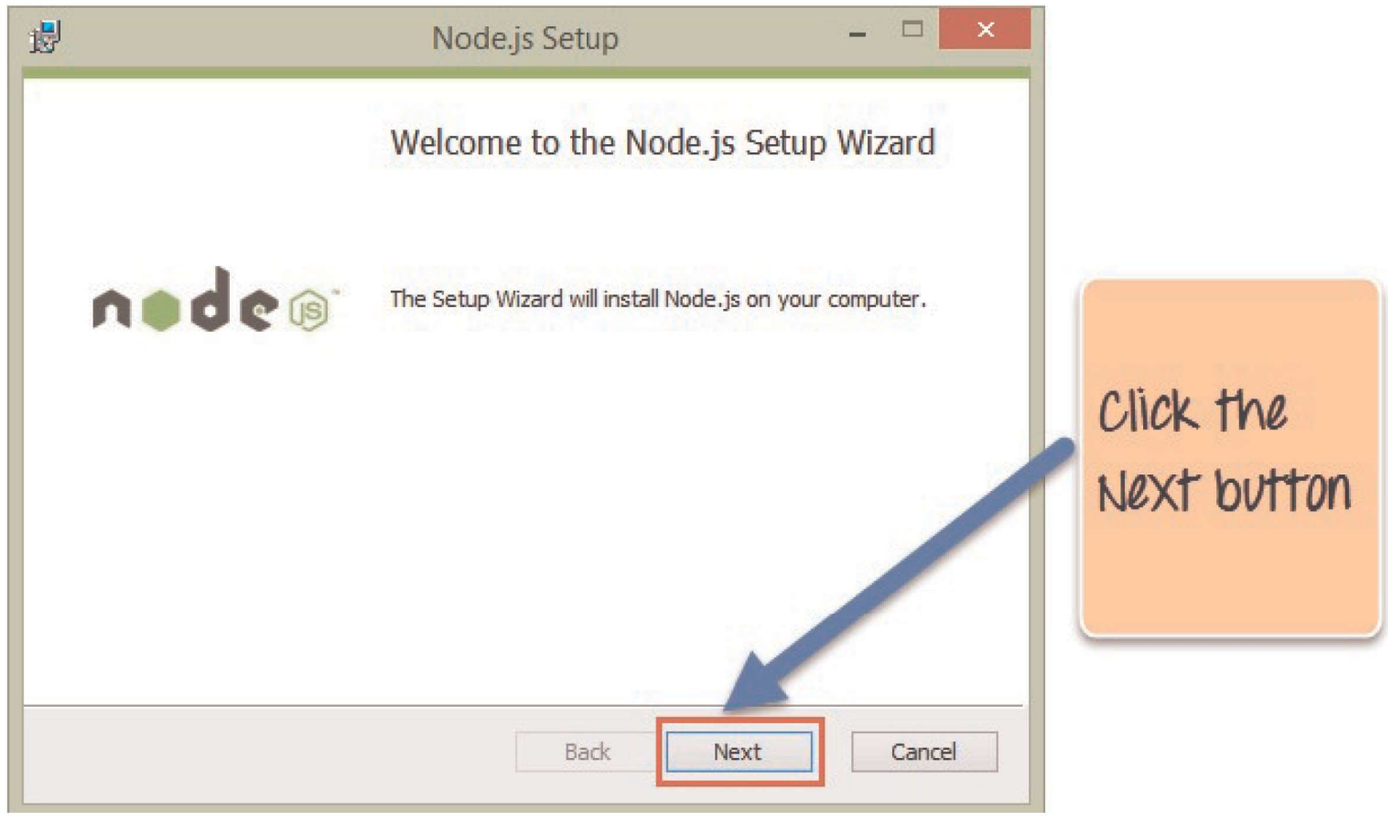
Download the Node.js source code or a pre-built installer for your platform, and start developing today.

Windows Installer (.msi)	32-bit	64-bit	
Windows Binary (.zip)	32-bit	64-bit	
macOS Installer (.pkg)	64-bit		
macOS Binary (.tar.gz)	64-bit		
Linux Binaries (x64)	64-bit		
Linux Binaries (ARM)	ARMv6	ARMv7	ARMv8
Source Code	node-v10.16.3.tar.gz		

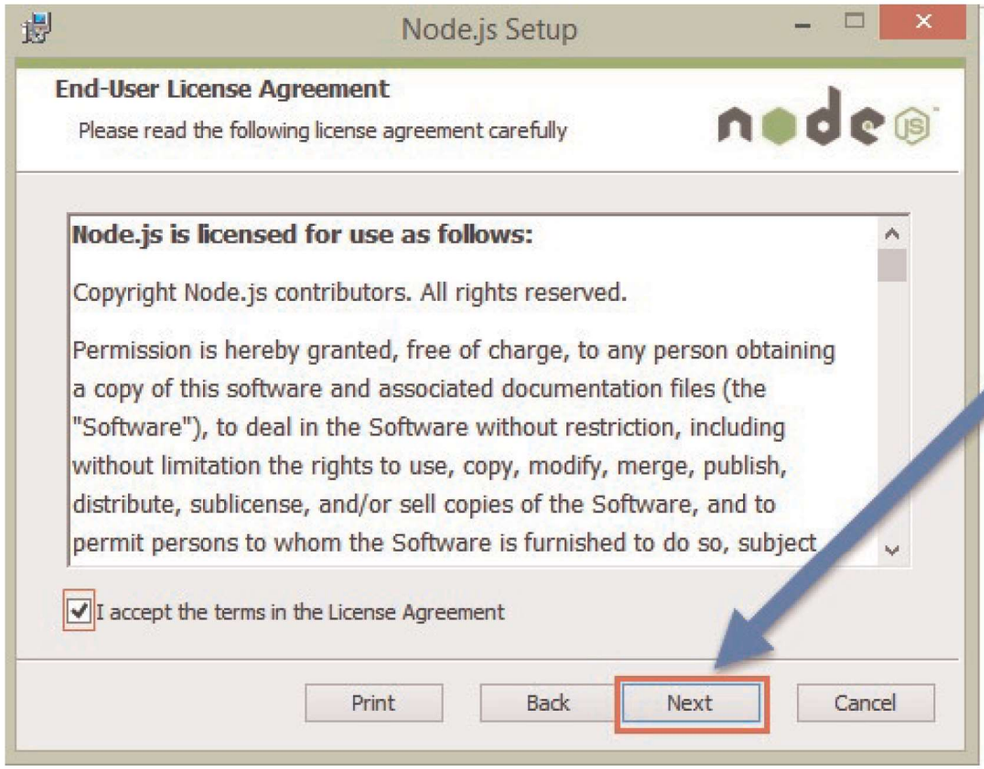
Step 2 :


Open File - Security Warning
Do you want to run this file?
Name: C:\All Downloads\Chrome\node-v4.2.3-x86.msi
Publisher: Node.js Foundation
Type: Windows Installer Package
From: C:\All Downloads\Chrome\node-v4.2.3-x86.msi
Run
Cancel
Always ask before opening this file
While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust.
What's the risk?
Click the Run button

Step 3 :

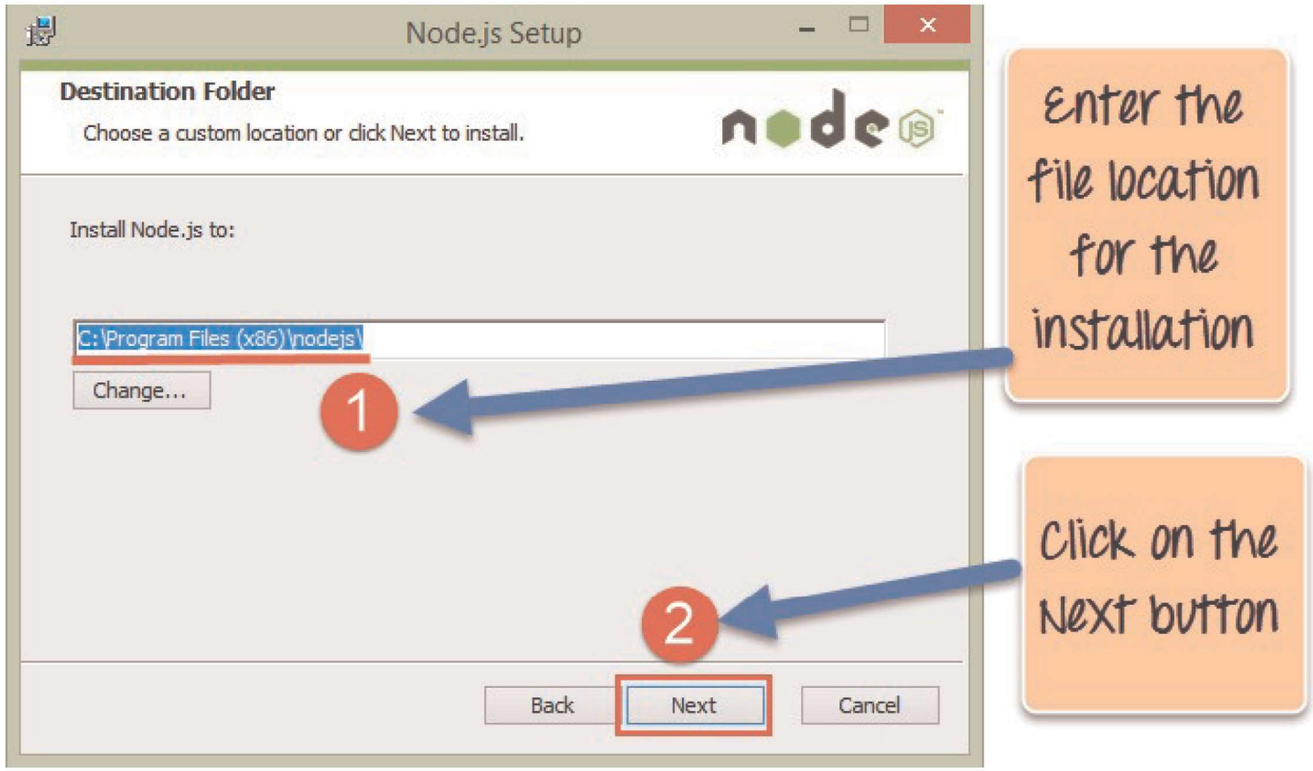

Node.js Setup
Welcome to the Node.js Setup Wizard
The Setup Wizard will install Node.js on your computer.
Back
Next
Cancel
Click the Next button

Step 4 :

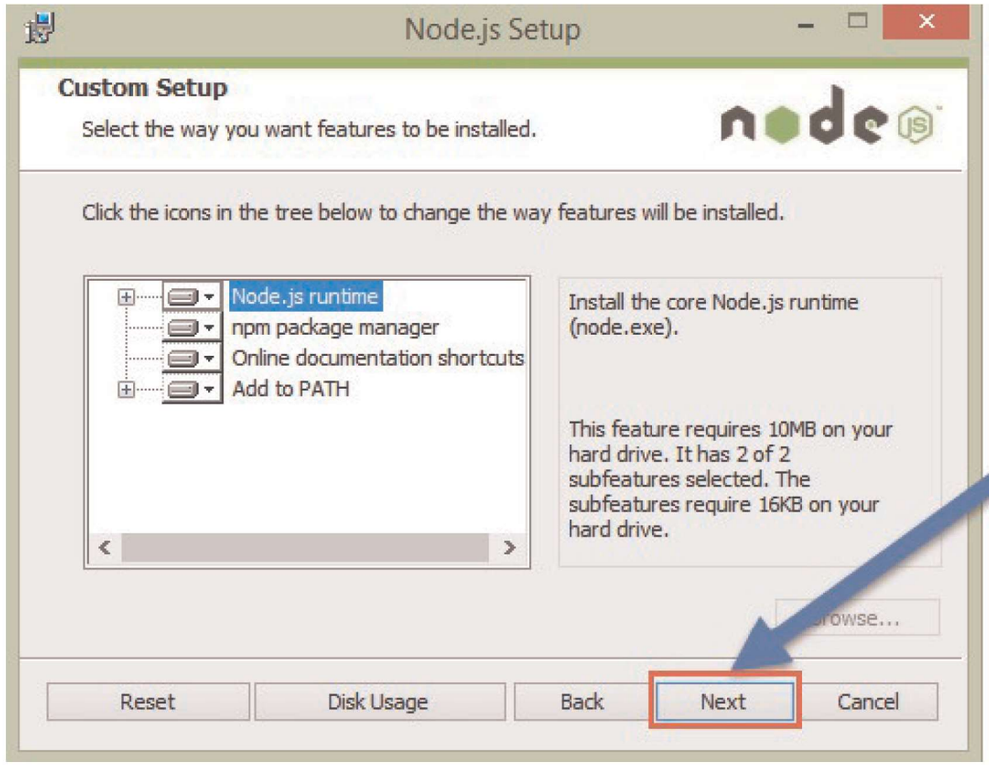


Accept the license agreement and click the Next

Step 5 :

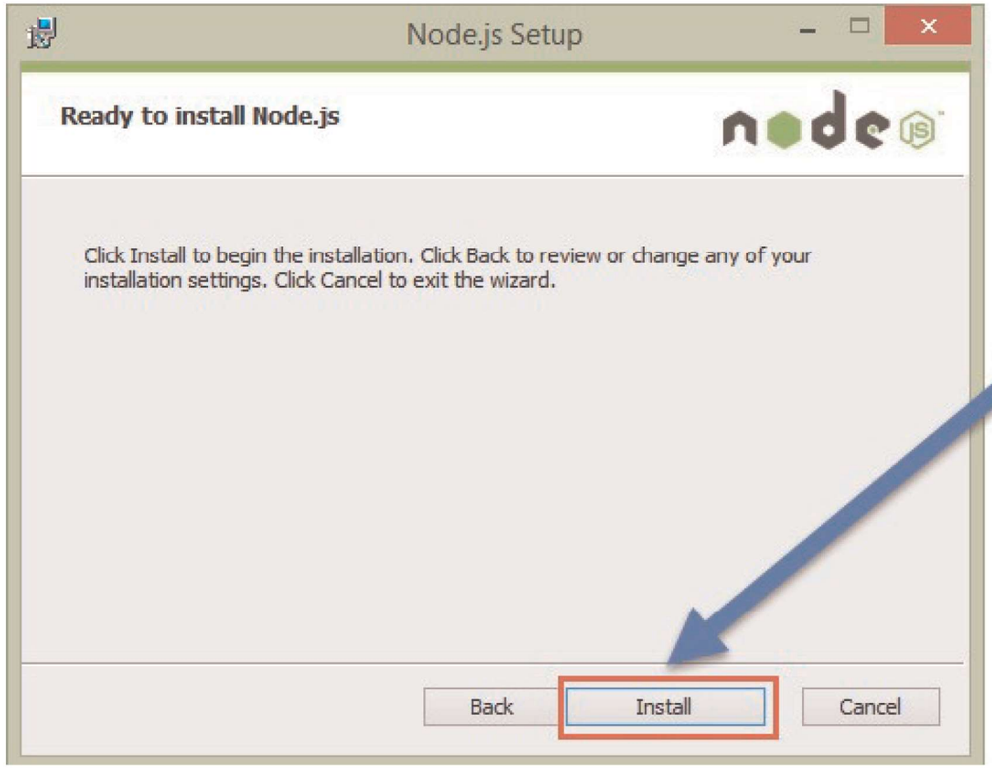

Node.js Setup
Destination Folder
Choose a custom location or click Next to install.
Install Node.js to:
C:\Program Files (x86)\nodejs\
Change...
1
2
Back
Next
Cancel
Enter the file location for the installation
Click on the Next button

Step 6 :



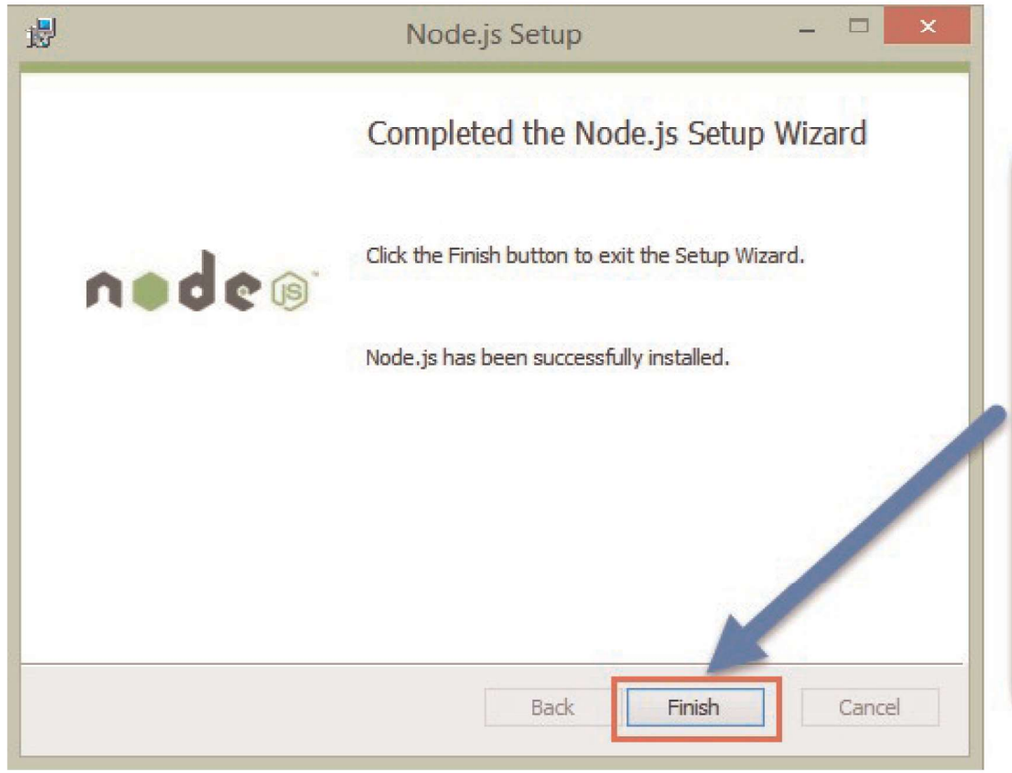
Accept the default components and click on Next

Step 7 :



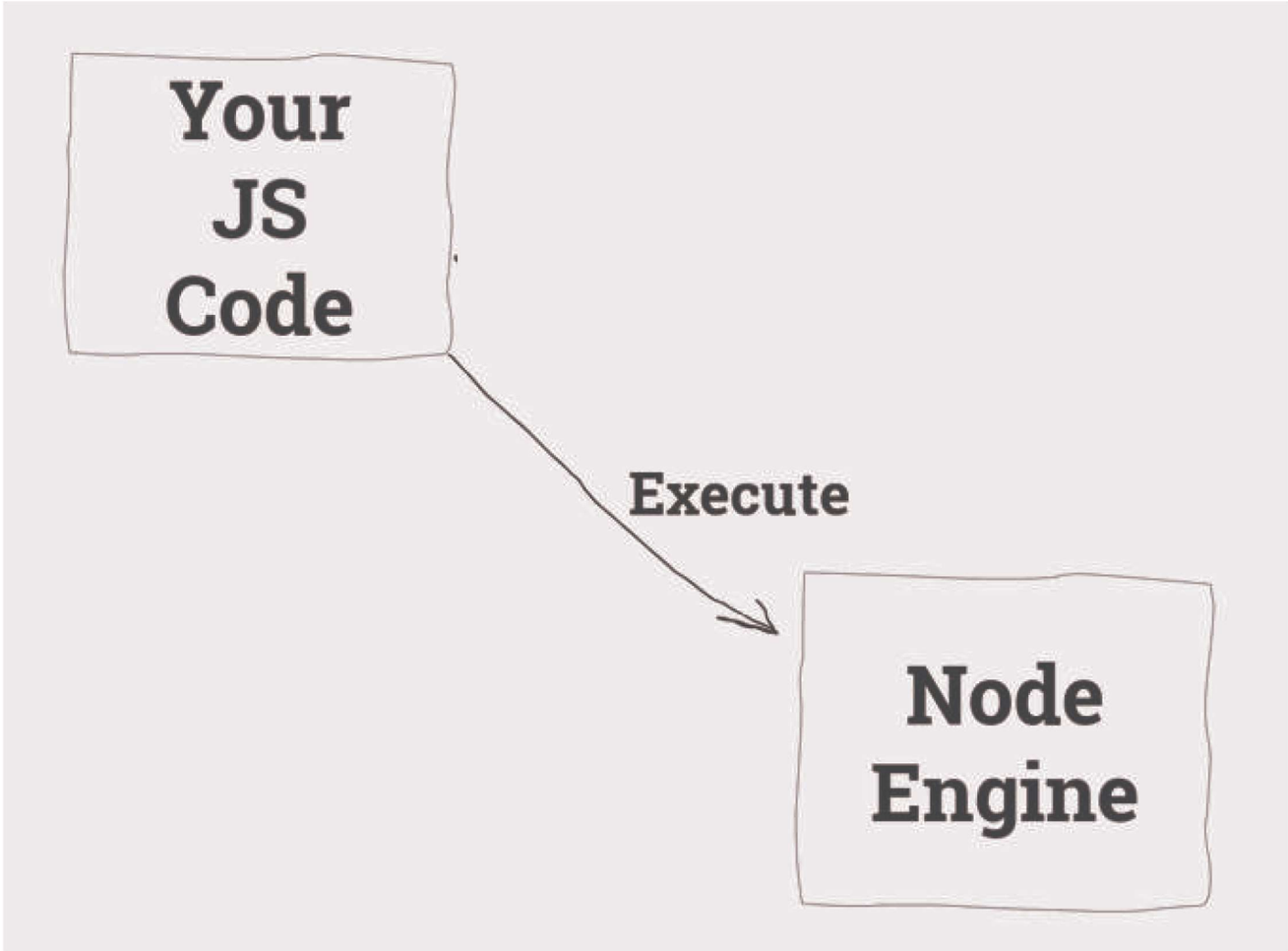
Click the Next button to begin the installation

Step 8 :

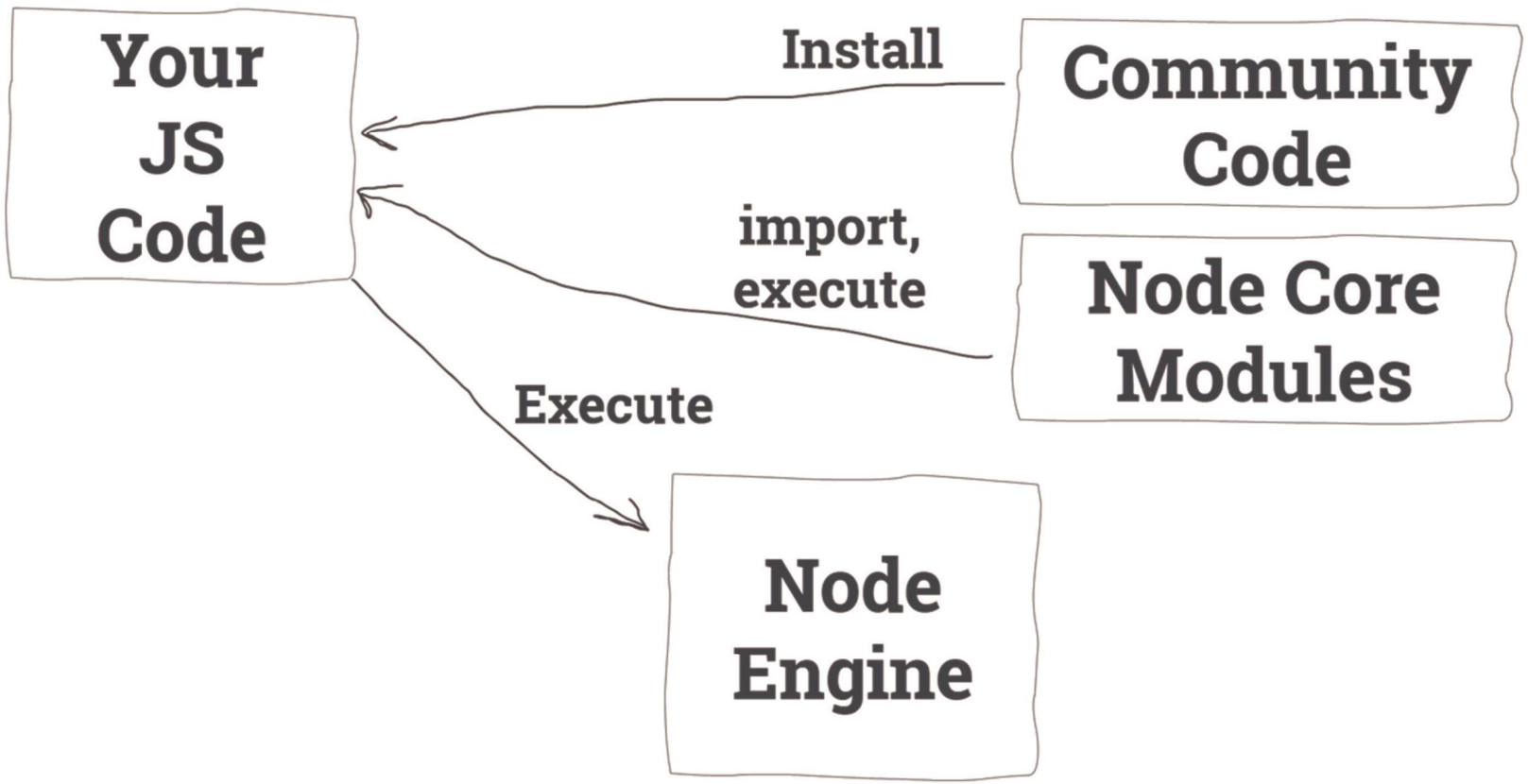


Click the Finish button to complete the installation

The Basic Pillars - JS



The Basic Pillars - Modules



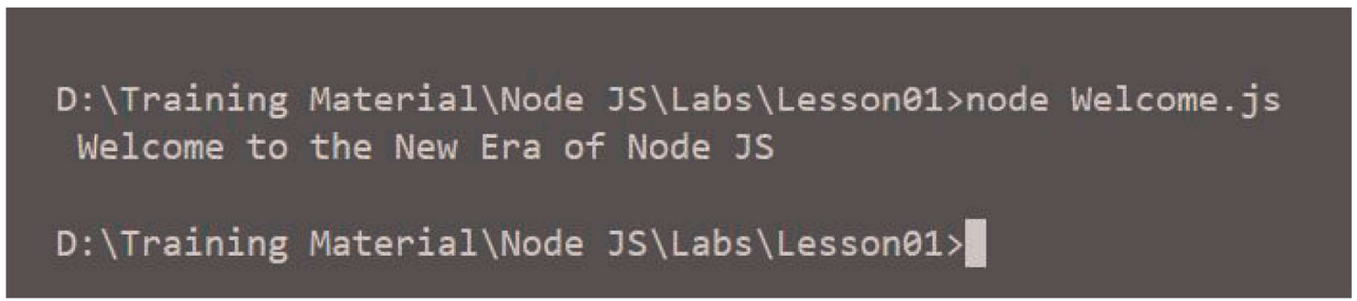
First Console Application (Welcome.js)

Welcome.js

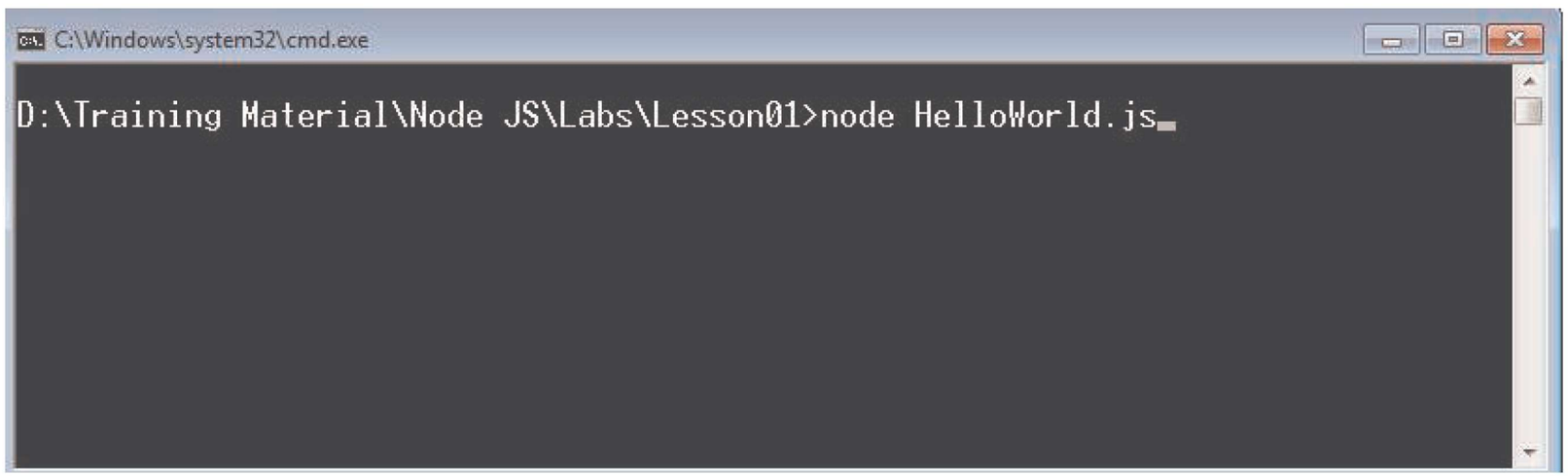
```
console.log('Welcome to the New Era of Node JS ');
```

Open Node.js command prompt and run the following code:

```
node Welcome.js
```



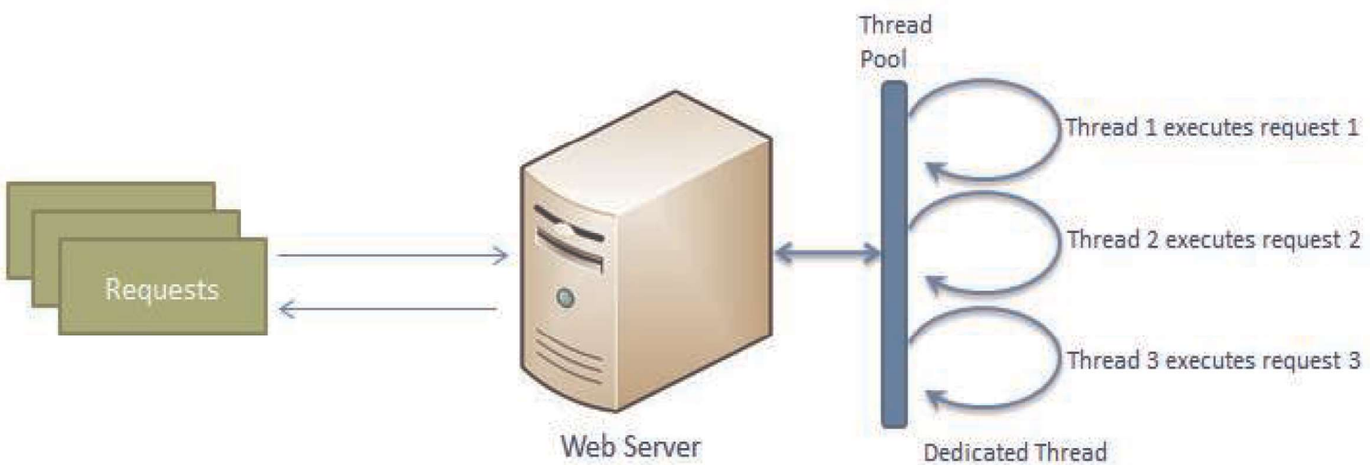
Running your first Hello World Web Application in Node.js



Node.js Process Model

Traditional Web Server Model

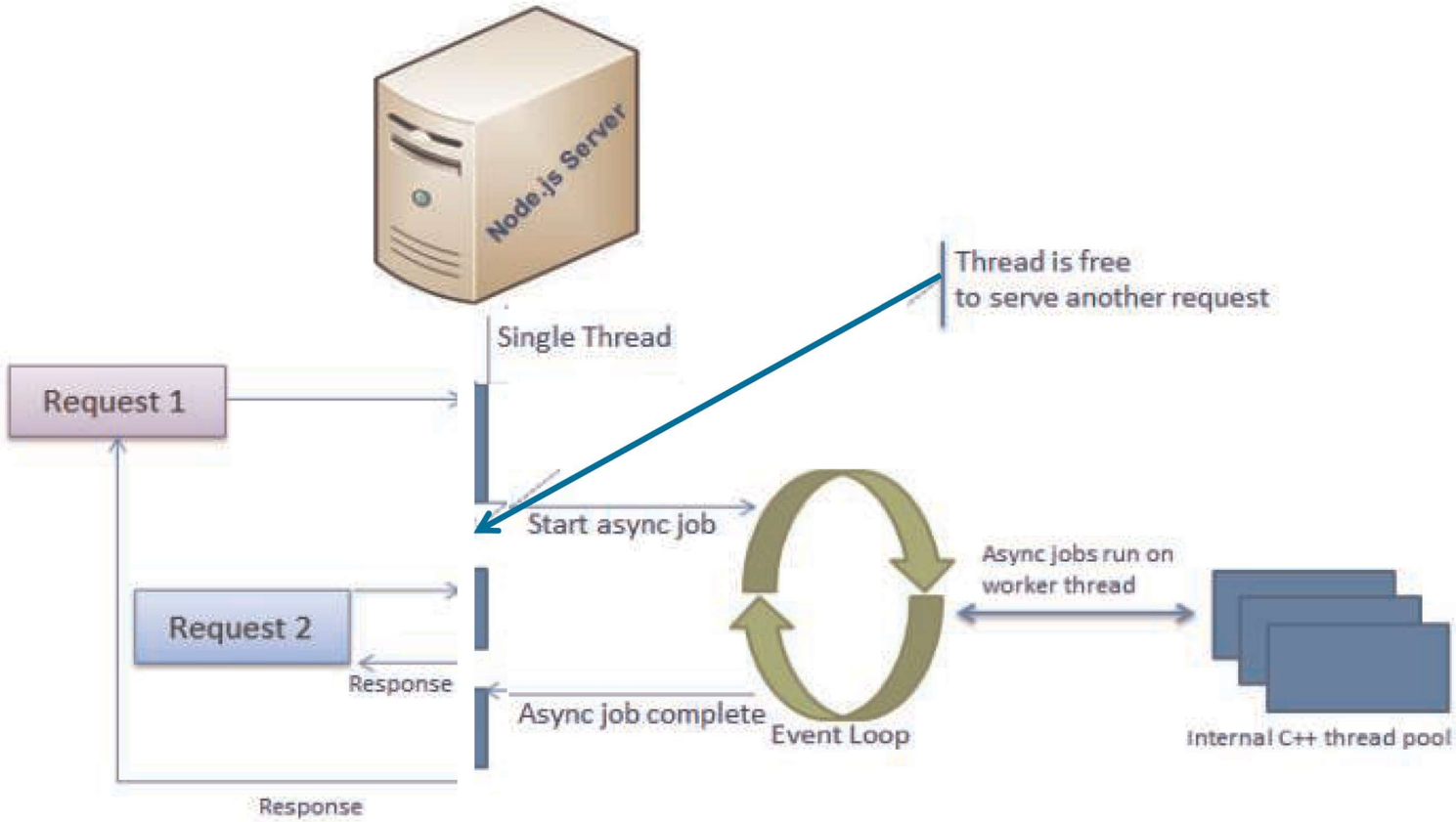
- In the traditional web server model, each request is handled by a dedicated thread from the thread pool.
- If no thread is available in the thread pool at any point of time then the request waits till the next available thread.
- Dedicated thread executes a particular request and does not return to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

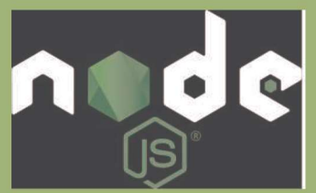


Node.js Process Model

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

Node.js Process Model





Node.js Console REPL

Node.js Console

There are three console methods that are used to write any node.js stream:

1. console.log()
2. console.error()
3. console.warn()

Node.js console.error()

The console.error() function is used to render error message on console.

```
console.error(new Error('Hell! This is a wrong method.'));
```

Open Node.js command prompt and run the following code:

```
node Console_error.js
```

```
D:\Training Material\Node JS\Labs\Lesson01>node Console_error.js
Error: Hell! This is a wrong method.
    at Object.<anonymous> (D:\Training Material\Node JS\Labs\Lesson01\Console_error.
js:1:15)
    at Module._compile (internal/modules/cjs/loader.js:776:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:787:10)
    at Module.load (internal/modules/cjs/loader.js:653:32)
    at tryModuleLoad (internal/modules/cjs/loader.js:593:12)
    at Function.Module._load (internal/modules/cjs/loader.js:585:3)
    at Function.Module.runMain (internal/modules/cjs/loader.js:829:12)
    at startup (internal/bootstrap/node.js:283:19)
    at bootstrapNodeJSCore (internal/bootstrap/node.js:622:3)

D:\Training Material\Node JS\Labs\Lesson01>
```

Node.js console.warn()

The console.warn() function is used to render error message on console.

```
const name = ‘Rahul';
console.warn(`Don't mess with me ${name}! Don't mess with me!`);
```

Open Node.js command prompt and run the following code:

```
node Console_warn.js
```

REPL

- Node.js comes with virtual environment called REPL (Node shell).
- **REPL stands for Read-Eval-Print-Loop.**
- It is a quick and easy way to test simple Node.js/JavaScript code.
- To launch the REPL (Node shell), open command prompt (in Windows) or terminal (in Mac or UNIX/Linux)

Node.js/JavaScript expression in REPL.

```
> 10 + 20
30
```

The + operator also concatenates strings as in browser's JavaScript.

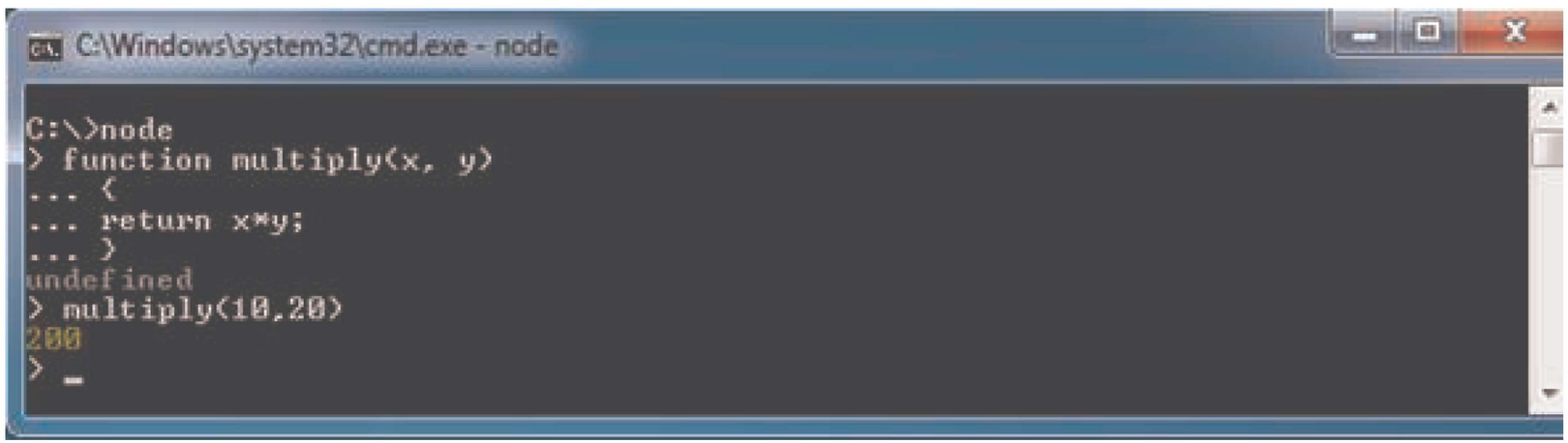
```
> "Hello" + "World"
Hello World
```

You can also define variables and perform some operation on them.

```
> var x = 10, y = 20;
> x + y
30
```

Cont …

- If you need to write multi line JavaScript expression or function then just press **Enter** whenever you want to write something in the next line as a continuation of your code.
- The REPL terminal will display three dots (...), it means you can continue on next line. Write .break to get out of continuity mode.



- To exit from the REPL terminal, press Ctrl + C twice or write .exit and press Enter.

- Thus, you can execute any Node.js/JavaScript code in the node shell (REPL). This will give you a result which is similar to the one you will get in the console of Google Chrome browser.

Using variable

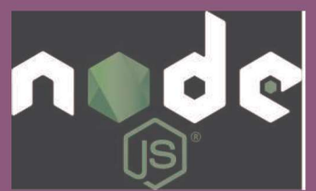
- Variables are used to store values and print later.
- If you don't use **var** keyword then value is stored in the variable and printed whereas if **var** keyword is used then value is stored but not printed.
- You can print variables using console.log().

Node.js Underscore Variable

- You can also use underscore _ to get the last result.

REPL Commands

REPL Command	Description
.help	Display help on all the commands
tab Keys	Display the list of all commands.
Up/Down Keys	See previous commands applied in REPL.
.save filename	Save current Node REPL session to a file.
.load filename	Load the specified file in the current Node REPL session.
ctrl + c	Terminate the current command.
ctrl + c (twice)	Exit from the REPL.
ctrl + d	Exit from the REPL.
.break	Exit from multiline expression.
.clear	Exit from multiline expression.



Node.js Basics

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

Primitive Types

- String
- Number
- Boolean
- Undefined
- Null
- RegExp

Everything else is an object in Node.js.

Loose Typing and Object Literal

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

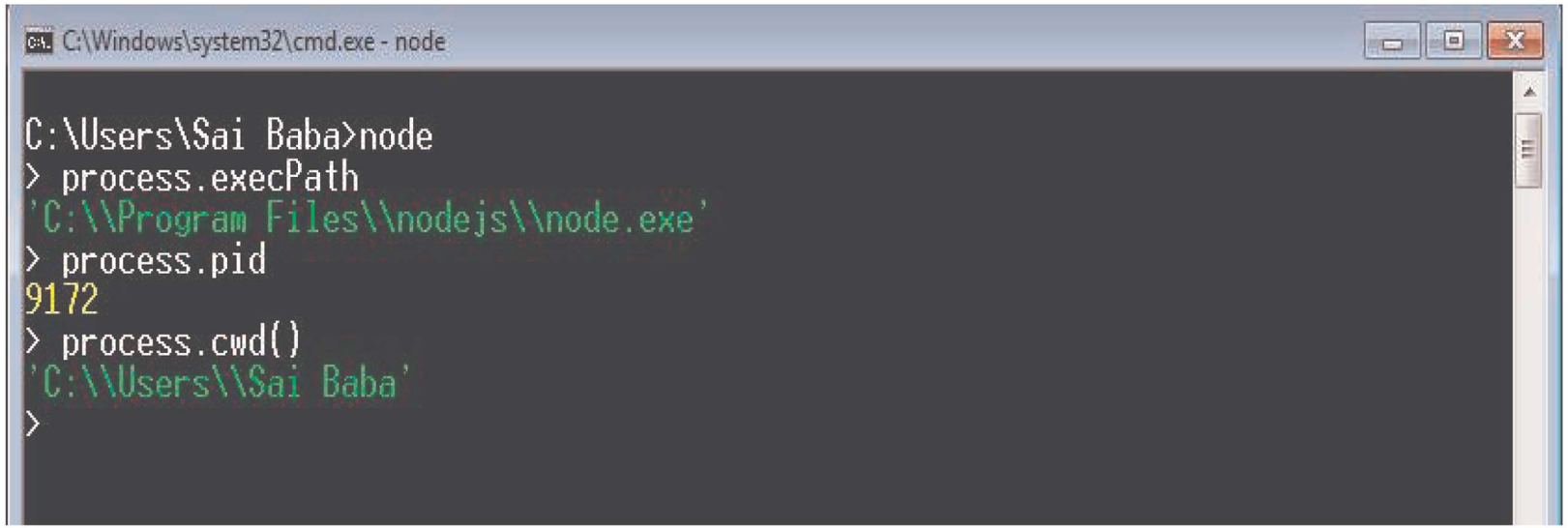
Object Literal

Example: Object

```
var obj = {
    authorName: 'Ryan Dahl',
    language: 'Node.js'
}
```

Process object

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



Function	Description
cwd()	returns path of current working directory
hrtime()	returns the current high-resolution real time in a [seconds, nanoseconds] array
memoryUsage()	returns an object having information of memory usage.
process.kill(pid[, signal])	is used to kill the given pid.
uptime()	returns the Node.js process uptime in seconds.

Defaults to local

- Node's JavaScript is different from browser's JavaScript when it comes to global scope.

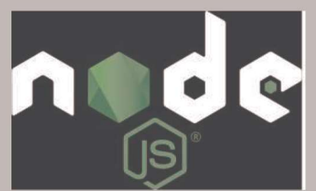
- In the browser's JavaScript, variables declared without var keyword become global. In Node.js, everything becomes local by default.

Access Global Scope

- In a browser, global scope is the window object. In Node.js, **global** object represents the global scope.
- To add something in global scope, you need to export it using export or module.export. The same way, import modules/object using require() function to access it from the global scope.

Example:

```
exports.log = {
    console: function(msg) {
        console.log(msg);
    },
    file: function(msg) {
        // log to file here
      }
}
```



Node.js Modules

Node.js Module

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

- That's where modules in Node.js comes in action.

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

Node.js Module Types

Node.js includes three types of modules:

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

Node.js Core Modules

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

Core Module	Description
http	http module includes classes, methods and events to create Node.js http server.
url	url module includes methods for URL resolution and parsing.
querystring	querystring module includes methods to deal with query string.
path	path module includes methods to deal with file paths.
fs	fs module includes classes, methods, and events to work with file I/O.
util	util module includes utility functions useful for programmers.

Loading Core Modules

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

```
var module = require('module_name');
```

- The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

Sample Code

- require() function returns an object because http module returns its functionality as an object, you can then use its properties and methods using dot notation e.g. http.createServer().
- In this way, you can load and use Node.js core modules in your application.

Example: Load and Use Core http Module

```
var http = require('http');

var server = http.createServer(function(req, res){

  //write code here

});

server.listen(5000);
```

Node.js Local Module

- Local modules are modules created locally in your Node.js application.
- These modules include different functionalities of your application in separate files and folders.
- You can also package it and distribute it via NPM, so that Node.js community can use it.
- For example, if you need to connect to MongoDB and fetch data then you can create a module for it, which can be reused in your application.

Writing Simple Module

- Simple logging module which logs the information, warning or error to the console.

Log.js

```
var log = {
            info: function (info) {
                console.log('Info: ' + info);
            },
            warning:function (warning) {
                console.log('Warning: ' + warning);
            },
            error:function (error) {
                console.log('Error: ' + error);
            }
    };

module.exports = log
```

- The *module.exports* is a special object which is included in every JS file in the Node.js application by default. Use **module.exports** or **exports** to expose a function, object or variable as a module in Node.js.

Loading Local Module

- To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.

app.js

```
var myLogModule = require('./Log.js');

myLogModule.info('Node.js started');
```

data.js

```
module.exports = {
    firstName: 'James',
    lastName: 'Bond'
}
```

app.js

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

Run the above example and see the result as shown below.

```
C:\> node app.js
James Bond
```

- Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the exportforms.
- Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the import forms.

Exporting and Importing

```
//exporter.ts
var sayHi = function(): void {
    console.log("Hello!");
}

export = sayHi;
```

```
//importer.ts
import sayHi = require('./exporter');
sayHi();
```

Exporting a declaration

- Any declaration (such as a variable, function, class, type alias, or interface) can be exported by adding the export keyword.

```
//Validation.ts
export interface StringValidator
{
    isAcceptable(s: string): boolean;
}
export = sayHi;
```

Exporting Variable and Class

```
//ZipCodeValidator.ts
export const numberRegexp = /^[0-9]+$/;
export class ZipCodeValidator implements StringValidator
    {
        isAcceptable(s: string) { return s.length === 5 && numberRegexp.test(s);
     }
}
```

Export statements

```
class ZipCodeValidator implements StringValidator {
    isAcceptable(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }
}
export { ZipCodeValidator };
export { ZipCodeValidator as mainValidator };
```

Import

- Importing is just about as easy as exporting from a module. Importing an exported declaration is done through using one of the import forms below:

Import a single export from a module

```
import { ZipCodeValidator } from "./ZipCodeValidator";

let myValidator = new ZipCodeValidator();
```

imports can also be renamed

```
import { ZipCodeValidator as ZCV } from "./ZipCodeValidator";
let myValidator = new ZCV();
```

Import the entire module into a single variable, and use it to access the module exports

```
import * as validator from "./ZipCodeValidator";
let myValidator = new validator.ZipCodeValidator();
```

What are Third Party modules in Node.js?

There are many readymade modules available in the market which can be used within Node js.

1. **Express framework** – Express is a minimal and flexible Node js web application framework that provides a robust set of features for the web and mobile applications.
2. **Socket.io** - Socket.IO enables real-time bidirectional event-based communication. This module is good for creation of chatting based applications.
3. **Jade** - Jade is a high-performance template engine and implemented with JavaScript for node and browsers.
4. **MongoDB** - The MongoDB Node.js driver is the officially supported node.js driver for MongoDB.

5. **Restify** - restify is a lightweight framework, similar to express for building REST APIs
6. **Bluebird** - Bluebird is a fully-featured promise library with a focus on innovative features and performance

Node Package Manager (npm)

Node Package Manager

- Node Package Manager (NPM) is a command line tool that installs, updates or uninstalls Node.js packages in your application.
- It is also an online repository for open-source Node.js packages.
- The node community around the world creates useful modules and publishes them as packages in this repository.
- NPM is included with Node.js installation. After you install Node.js, verify NPM installation by writing the following command in terminal or command prompt.

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

```
C:\> npm install npm -g
```

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

```
C:\> npm help
```

- NPM performs the operation in two modes: global and local.
- In the global mode, NPM performs operations which affect all the Node.js applications on the computer whereas in the local mode, NPM performs operations for the particular local directory which affects an application in that directory only.

Install Package Locally

- Command to install any third party module in your local Node.js project folder.

```
C:\>npm install <package name>
```

- The following command will install ExpressJS into MyNodeProj folder.

```
C:\MyNodeProj> npm install express
```

- All the modules installed using NPM are installed under **node_modules** folder. The above command will create ExpressJS folder under node_modules folder in the root folder of your project and install Express.js there.

Add Dependency into package.json

- Use --save at the end of the install command to add dependency entry into package.json of your application.
- The following command will install ExpressJS in your application and also adds dependency entry into the package.json

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

The package.json of NodejsConsoleApp project

package.json

```
{
  "name": "NodejsConsoleApp",
  "version": "0.0.0",
  "description": "NodejsConsoleApp",
  "main": "app.js",
  "author": {
    "name": "Dev",
    "email": ""
  },
  "dependencies": {
    "express": "^4.13.3"
  }
}
```

Install Package Globally

- NPM can also install packages globally so that all the node.js application on that computer can import and use the installed packages. NPM installs global packages into */<User>/local/lib/node_modules* folder.
- Apply -g in the install command to install package globally.

```
C:\MyNodeProj> npm install -g express
```

Update Package

- To update the package installed locally in your Node.js project, navigate the command prompt or terminal window path to the project folder

```
C:\MyNodeProj> npm update <package name>
```

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

Uninstall Packages

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

```
C:\>npm uninstall <package name>
```

- The following command will uninstall ExpressJS from the application.

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

Searching a Module

- "npm search express" command is used to search express or module.

Node.js Global Objects (Implicit Objects)

- Node.js global objects are global in nature and available in all modules.
- You don't need to include these objects in your application; rather they can be used directly.
- These objects are modules, functions, strings and object etc. Some of these objects aren't actually in the global scope but in the module scope.

Cont …

1. __dirname
2. __filename
3. Console
4. Process
5. Buffer
6. setImmediate(callback[, arg][, ...])
7. setInterval(callback, delay[, arg][, ...])
8. setTimeout(callback, delay[, arg][, ...])
9. clearImmediate(immediateObject)
10. clearInterval(intervalObject)
11. clearTimeout(timeoutObject)

Node OS

Node.js OS

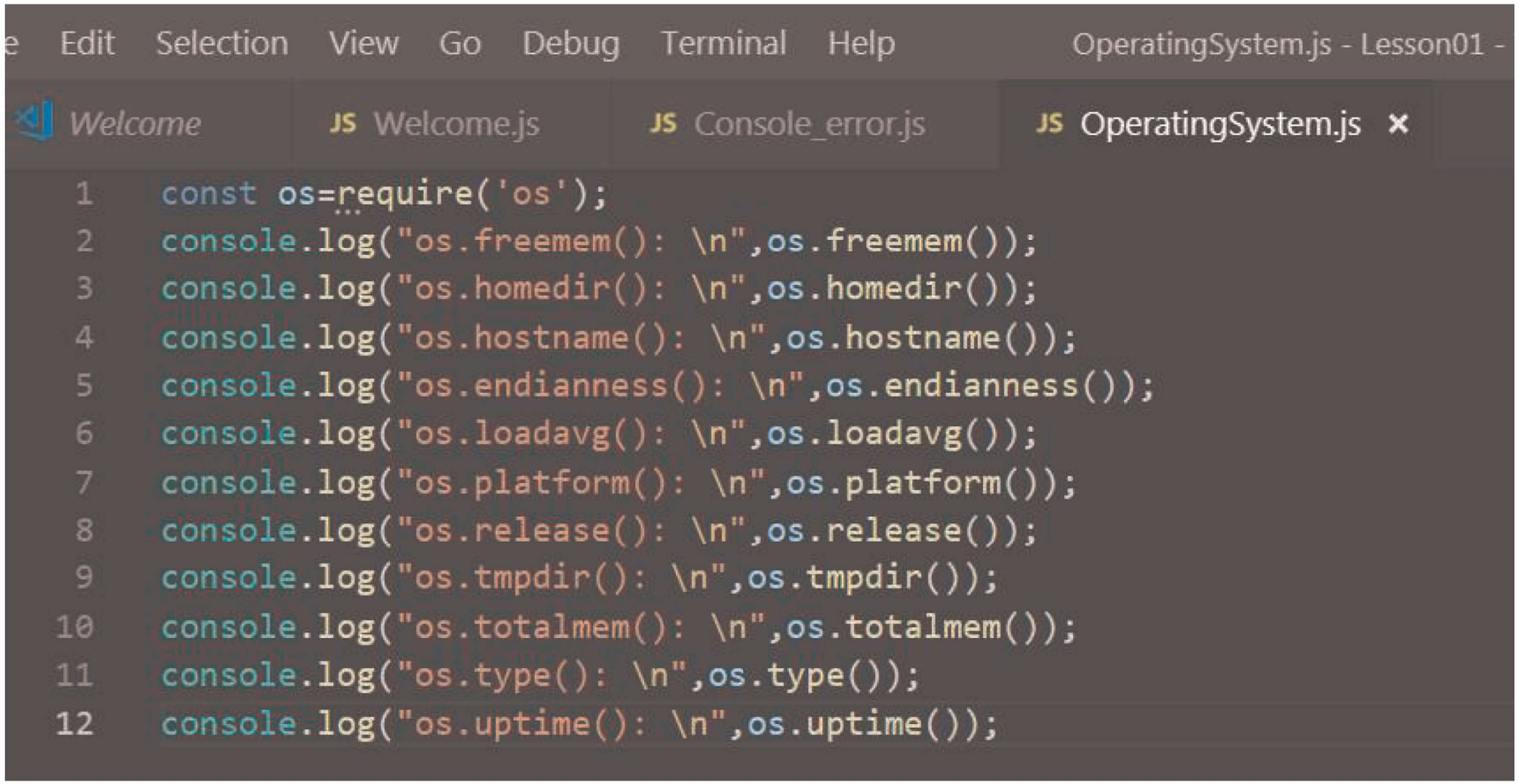
- Node.js OS provides some basic operating-system related utility functions. Let's see the list generally used functions or methods.

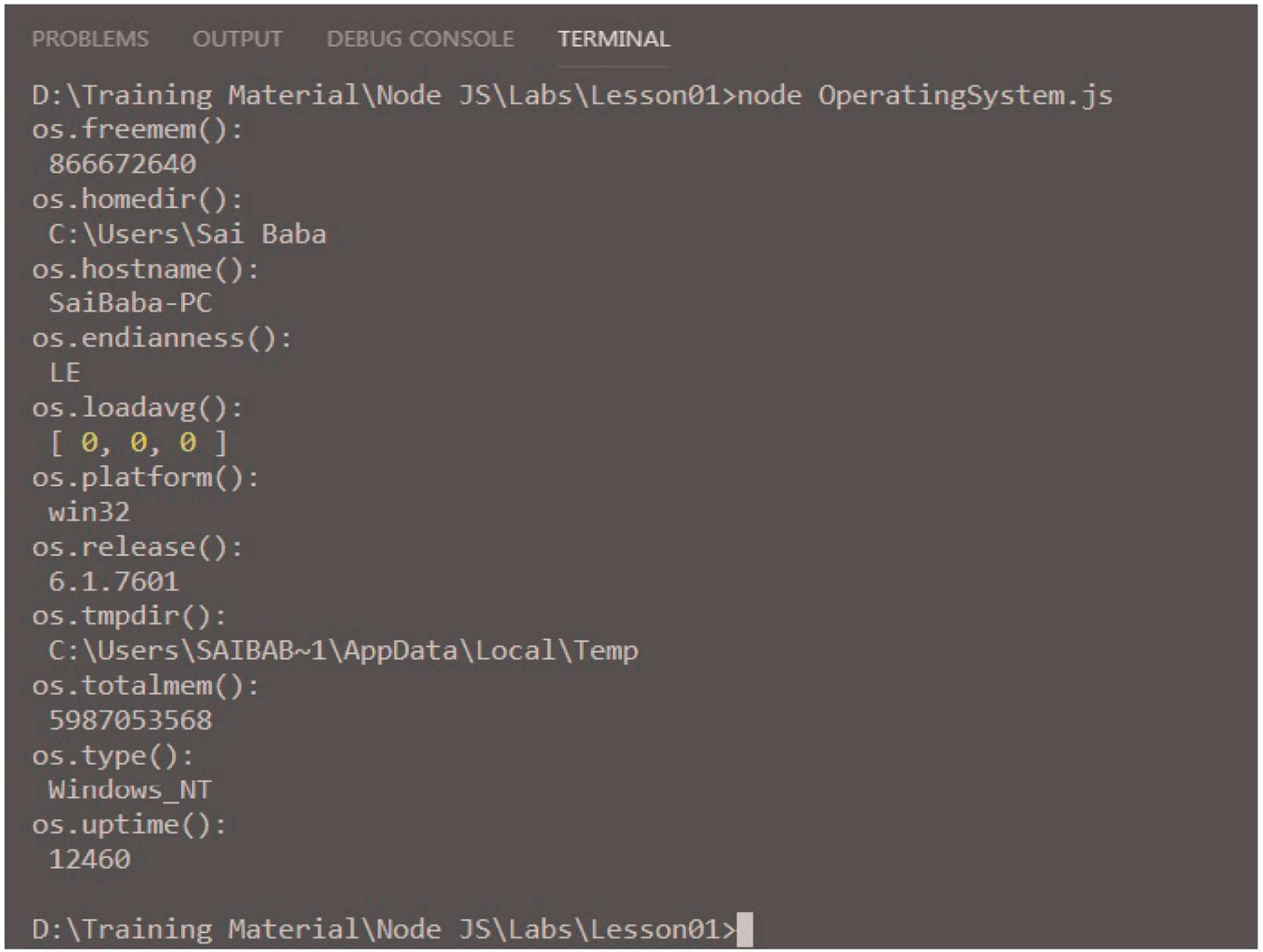
Index	Method	Description
1.	os.arch()	This method is used to fetch the operating system CPU architecture.
2.	os.cpus()	This method is used to fetch an array of objects containing information about each cpu/core installed: model, speed (in MHz), and times (an object containing the number of milliseconds the cpu/core spent in: user, nice, sys, idle, and irq).
3.	os.endianness()	This method returns the endianness of the cpu. Its possible values are 'BE' for big endian or 'LE' for little endian.
4.	os.freemem()	This methods returns the amount of free system memory in bytes.
5.	os.homedir()	This method returns the home directory of the current user.
6.	os.hostname()	This method is used to returns the hostname of the operating system.

Cont …

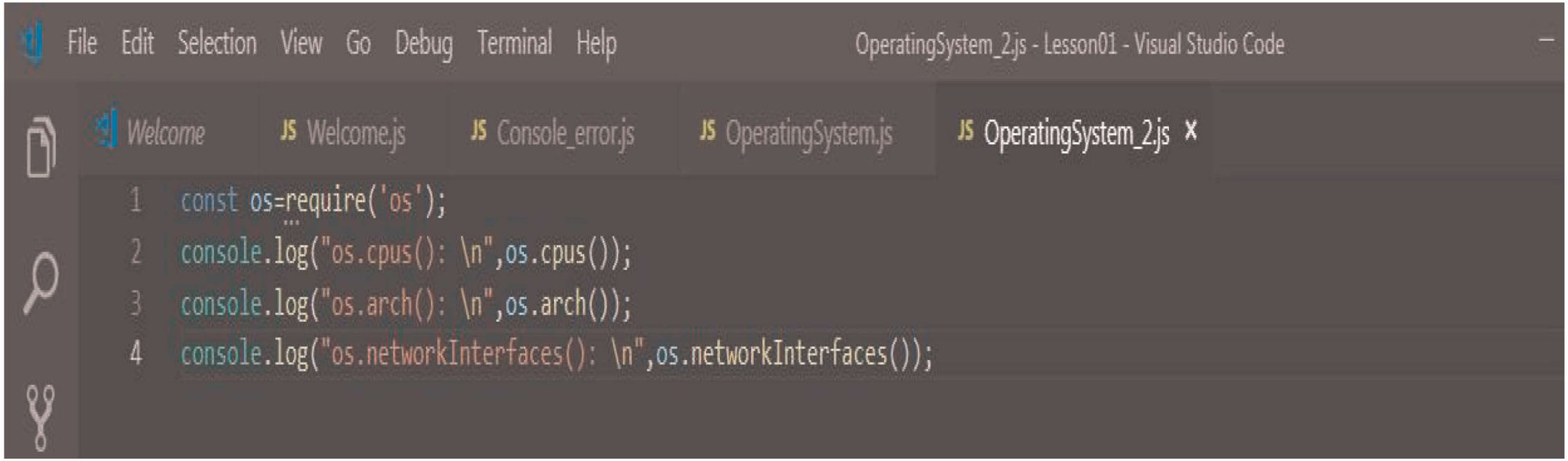
7.	os.loadavg()	This method returns an array containing the 1, 5, and 15 minute load averages. The load average is a time fraction taken by system activity, calculated by the operating system and expressed as a fractional number.
8.	os.networkinterfaces()	This method returns a list of network interfaces.
9.	os.platform()	This method returns the operating system platform of the running computer i.e.'darwin', 'win32','freebsd', 'linux', 'sunos' etc.
10.	os.release()	This method returns the operating system release.
11.	os.tmpdir()	This method returns the operating system's default directory for temporary files.
12.	os.totalmem()	This method returns the total amount of system memory in bytes.
13.	os.type()	This method returns the operating system name. For example 'linux' on linux, 'darwin' on os x and 'windows_nt' on windows.
14.	os.uptime()	This method returns the system uptime in seconds.
15.	os.userinfo([options])	This method returns a subset of the password file entry for the current effective user.

Sample

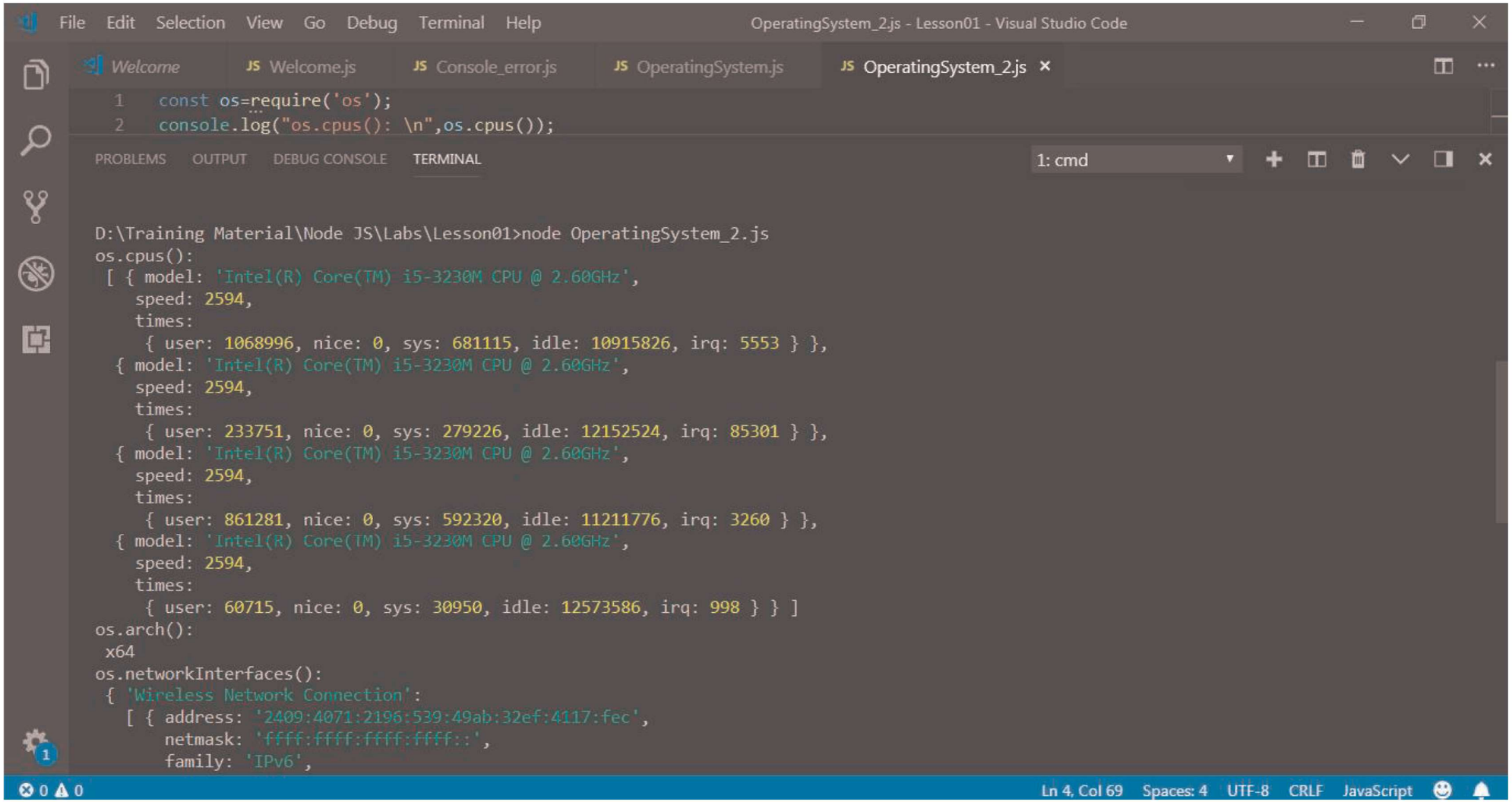

Edit Selection View Go Debug Terminal Help
OperatingSystem.js - Lesson01 -
Welcome
JS Welcome.js
JS Console_error.js
JS OperatingSystem.js
1 const os=require('os');
2 console.log("os.freemem(): \n",os.freemem());
3 console.log("os.homedir(): \n",os.homedir());
4 console.log("os.hostname(): \n",os.hostname());
5 console.log("os.endianness(): \n",os.endianness());
6 console.log("os.loadavg(): \n",os.loadavg());
7 console.log("os.platform(): \n",os.platform());
8 console.log("os.release(): \n",os.release());
9 console.log("os.tmpdir(): \n",os.tmpdir());
10 console.log("os.totalmem(): \n",os.totalmem());
11 console.log("os.type(): \n",os.type());
12 console.log("os.uptime(): \n",os.uptime());


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
D:\Training Material\Node JS\Labs\Lesson01>node OperatingSystem.js
os.freemem():
866672640
os.homedir():
C:\Users\Sai Baba
os.hostname():
SaiBaba-PC
os.endianness():
LE
os.loadavg():
[0, 0, 0]
os.platform():
win32
os.release():
6.1.7601
os.tmpdir():
C:\Users\SAIBAB~1\AppData\Local\Temp
os.totalmem():
5987053568
os.type():
Windows_NT
os.uptime():
12460
D:\Training Material\Node JS\Labs\Lesson01>

Sample 2


File Edit Selection View Go Debug Terminal Help
OperatingSystem_2.js - Lesson01 - Visual Studio Code
Welcome
Welcome.js
Console_error.js
OperatingSystem.js
OperatingSystem_2.js
1 const os=require('os');
2 console.log("os.cpus(): \n",os.cpus());
3 console.log("os.arch(): \n",os.arch());
4 console.log("os.networkInterfaces(): \n",os.networkInterfaces());

Output



Node Timers

Node.js Timer

- Node.js Timer functions are global functions. You don't need to use require() function in order to use timer functions.

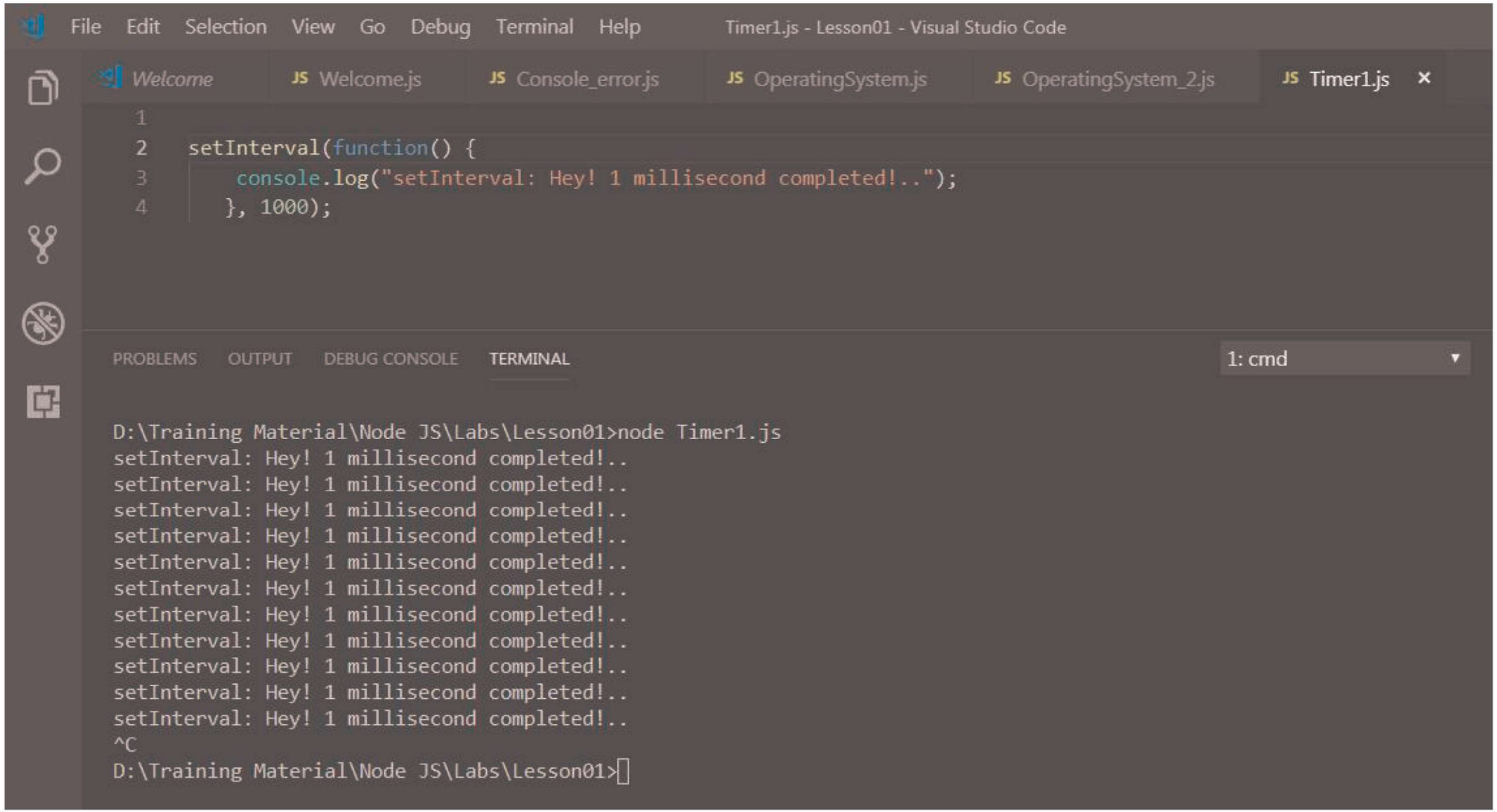
Set timer functions:

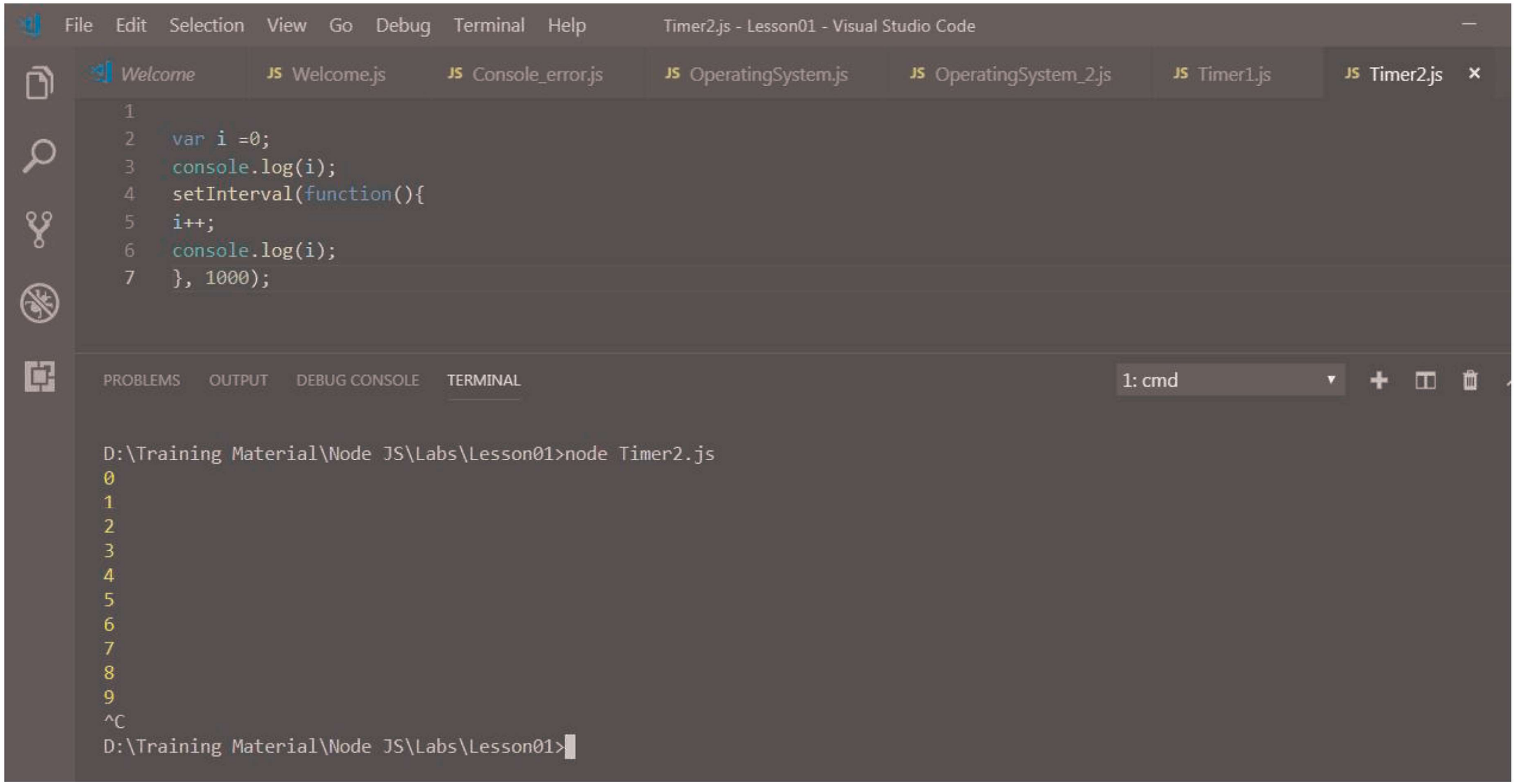
1. **setImmediate():** It is used to execute setImmediate.
2. **setInterval():** It is used to define a time interval.
3. **setTimeout():** ()- It is used to execute a one-time callback after delay milliseconds.

Clear timer functions:

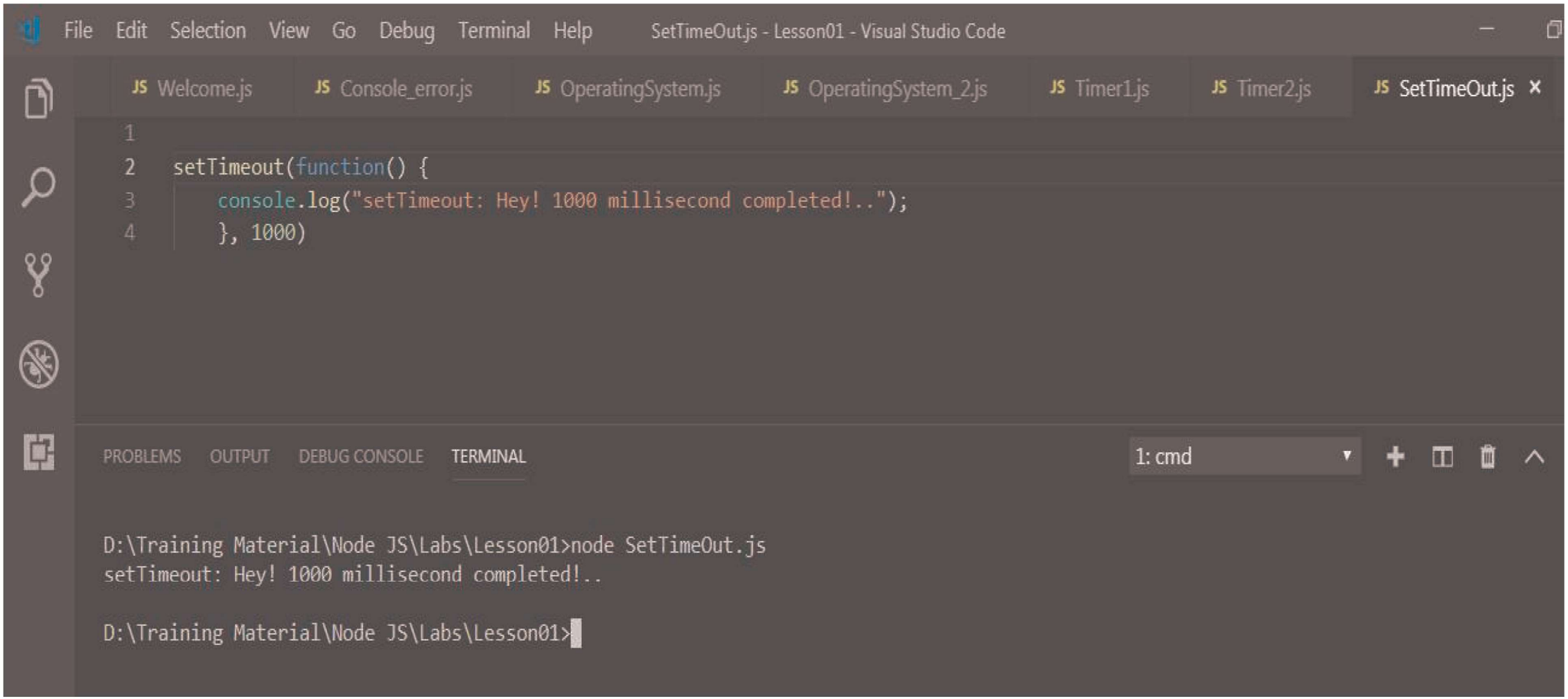
1. **clearImmediate(immediateObject):** It is used to stop an immediateObject, as created by setImmediate
2. **clearInterval(intervalObject):** It is used to stop an intervalObject, as created by setInterval
3. **clearTimeout(timeoutObject):** It prevents a timeoutObject, as created by setTimeout

Node.js Timer setInterval() Example




File Edit Selection View Go Debug Terminal Help
Timer2.js - Lesson01 - Visual Studio Code
Welcome
Welcome.js
Console_error.js
OperatingSystem.js
OperatingSystem_2.js
Timer1.js
Timer2.js
1
2 var i =0;
3 console.log(i);
4 setInterval(function(){
5 i++;
6 console.log(i);
7 }, 1000);
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
1: cmd
D:\Training Material\Node JS\Labs\Lesson01>node Timer2.js
0
1
2
3
4
5
6
7
8
9
^C
D:\Training Material\Node JS\Labs\Lesson01>

Node.js Timer setTimeout() Example



Node DNS

Node.js DNS

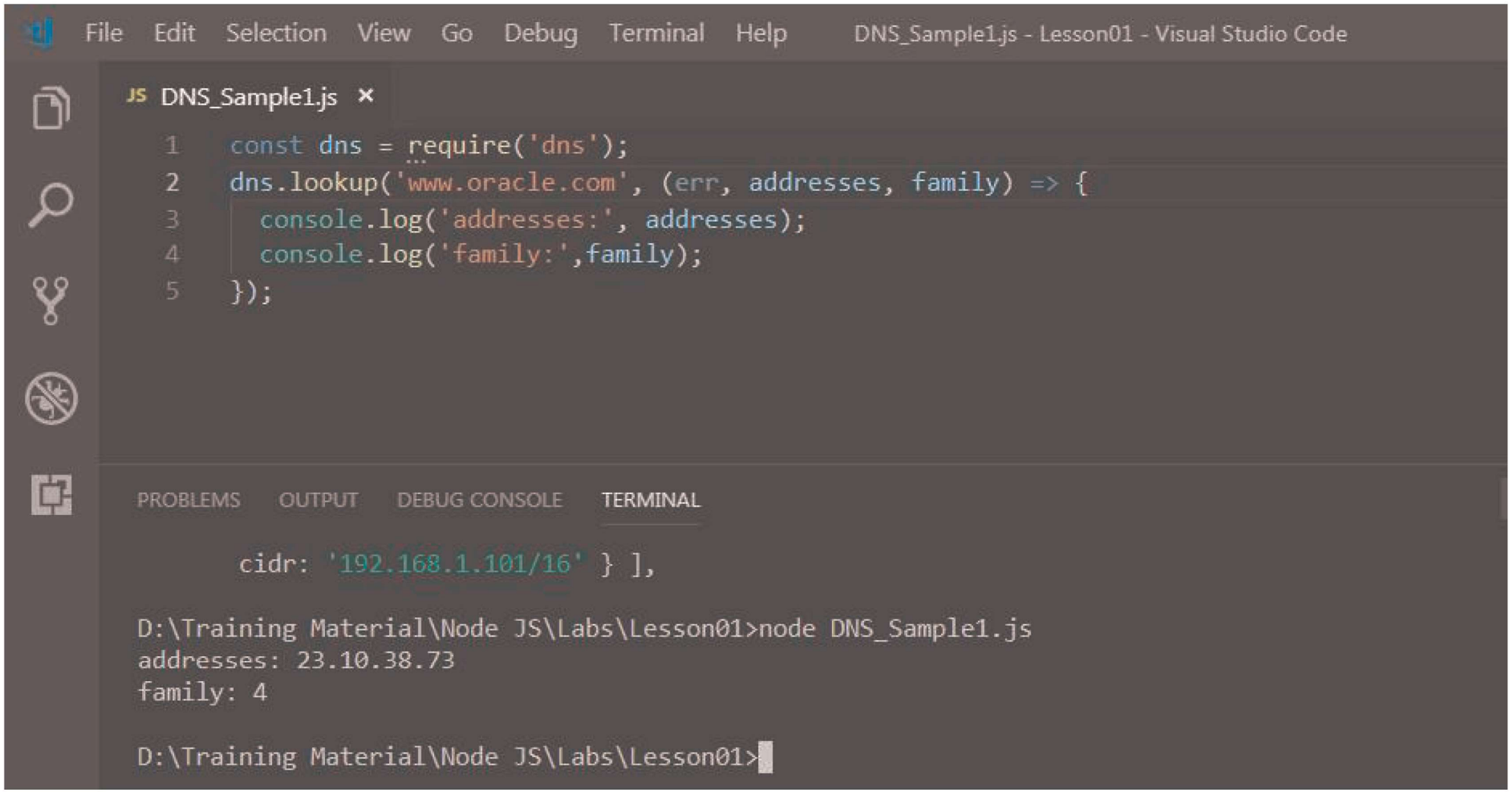
The Node.js DNS module contains methods to get information of given hostname.

1. dns.getServers()
2. dns.setServers(servers)
3. dns.lookup(hostname[, options], callback)
4. dns.lookupService(address, port, callback)
5. dns.resolve(hostname[, rrtype], callback)
6. dns.resolve4(hostname, callback)
7. dns.resolve6(hostname, callback)
8. dns.resolveCname(hostname, callback)

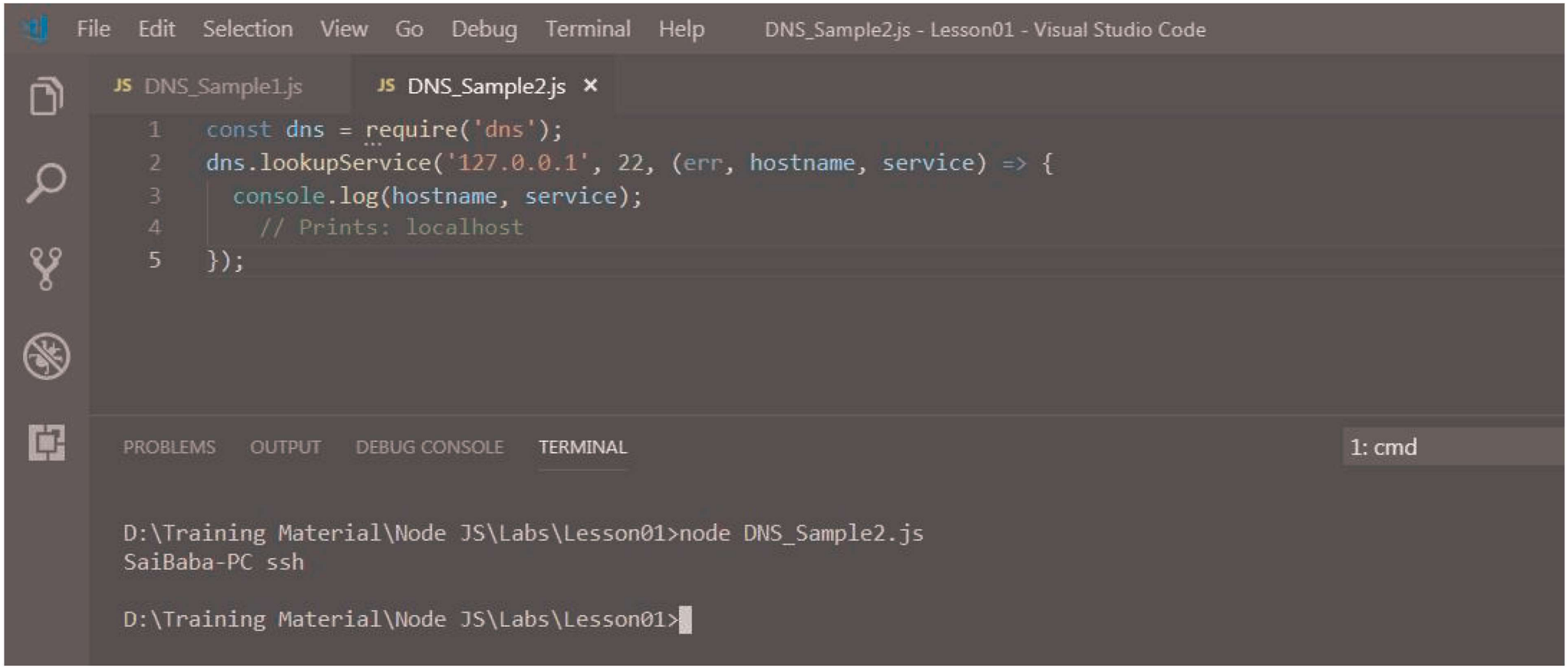
Cont …

9. dns.resolveMx(hostname, callback)
10. dns.resolveNs(hostname, callback)
11. dns.resolveSoa(hostname, callback)
12. dns.resolveSrv(hostname, callback)
13. dns.resolvePtr(hostname, callback)
14. dns.resolveTxt(hostname, callback)
15. dns.reverse(ip, callback)

Node.js DNS Example 1



DNS Sample 2


File Edit Selection View Go Debug Terminal Help
DNS_Sample2.js - Lesson01 - Visual Studio Code
DNS_Sample1.js
DNS_Sample2.js
1 const dns = require('dns');
2 dns.lookupService('127.0.0.1', 22, (err, hostname, service) => {
3 console.log(hostname, service);
4 // Prints: localhost
5 });
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: cmd
D:\Training Material\Node JS\Labs\Lesson01>node DNS_Sample2.js
SaiBaba-PC ssh
D:\Training Material\Node JS\Labs\Lesson01>

Node Buffers and Streams

Buffer

- Node.js includes an additional data type called Buffer (not available in browser's JavaScript).
- Node.js provides Buffer class to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap.
- Buffer class is used because pure JavaScript is not nice to binary data.
- Buffer is mainly used to store binary data, while reading from a file or receiving packets over the network.
- Buffer class is a global class. It can be accessed in application without importing buffer module.

Node.js Creating Buffers

- There are many ways to construct a Node buffer. Following are the three mostly used methods:

1. **Create an uninitiated buffer:** Following is the syntax of creating an uninitiated buffer of 10 octets:

```
var buf = new Buffer(10);
```

2. **Create a buffer from array:** Following is the syntax to create a Buffer from a given array:

```
var buf = new Buffer([10, 20, 30, 40, 50]);
```

3. **Create a buffer from string:** Following is the syntax to create a Buffer from a given string and optionally encoding type:

```
var buf = new Buffer("Simply Easy Learning", "utf-8");
```

- The **OCTET-STREAM** format is used for file attachments on the Web with an unknown file type. These .**octet-stream** files are arbitrary binary data files that may be in any multimedia format.

Node.js Writing to buffers

```
buf.write(string[, offset][, length][, encoding])
```

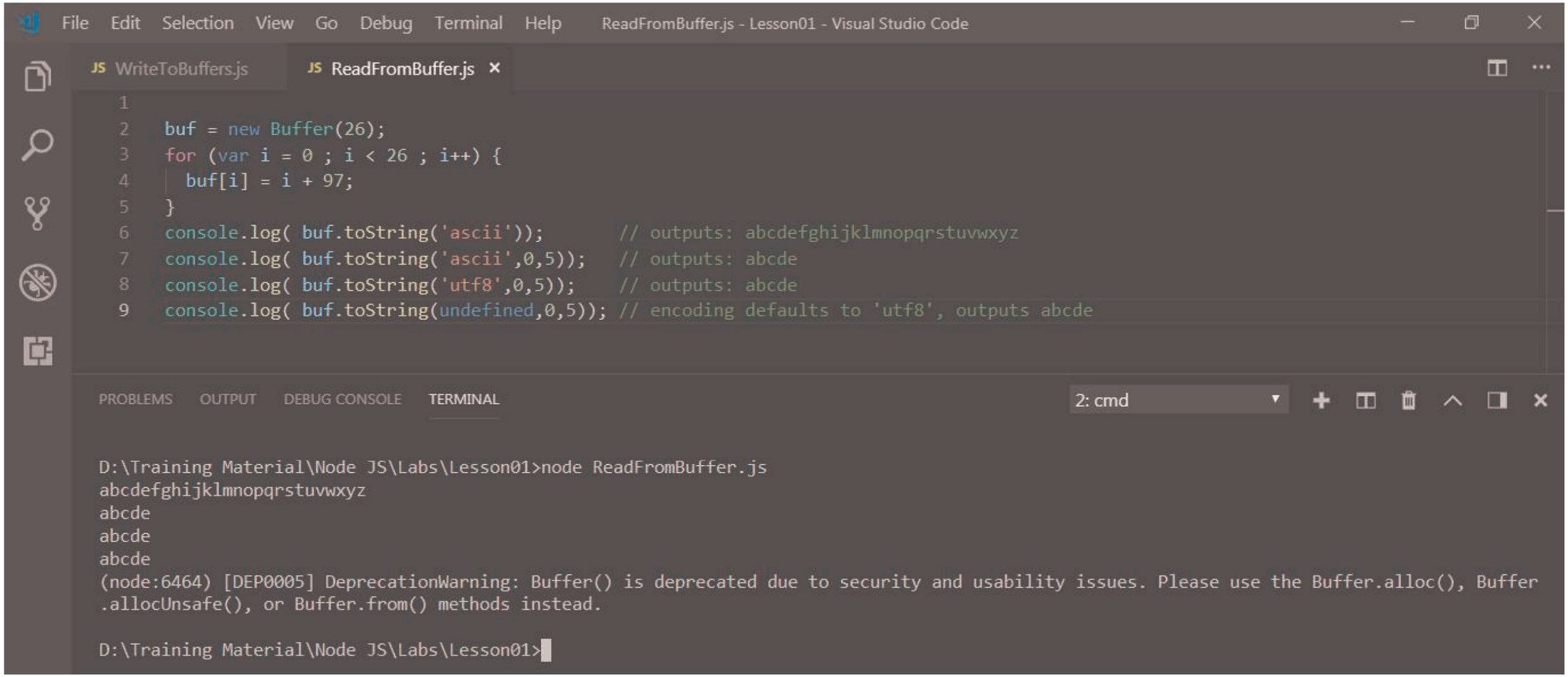
- **string:** It specifies the string data to be written to buffer.
- **offset:** It specifies the index of the buffer to start writing at. Its default value is 0.
- **length:** It specifies the number of bytes to write. Defaults to buffer.length
- **encoding:** Encoding to use. 'utf8' is the default encoding.


File Edit Selection View Go Debug Terminal Help
WriteToBuffers.js - Lesson01 - Visual Studio Code
WriteToBuffers.js
1 buf = new Buffer(256);
2 len = buf.write("Simply Easy Learning");
3 console.log("Octets written : "+ len);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: cmd
D:\Training Material\Node JS\Labs\Lesson01>node WriteToBuffers.js
Octets written : 20
(node:3500) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer
.allocUnsafe(), or Buffer.from() methods instead.
D:\Training Material\Node JS\Labs\Lesson01>

Node.js Reading from buffers

```
buf.toString([encoding][, start][, end])
```

- **encoding:** It specifies encoding to use. 'utf8' is the default encoding
- **start:** It specifies beginning index to start reading, defaults to 0.
- **end:** It specifies end index to end reading, defaults is complete buffer.

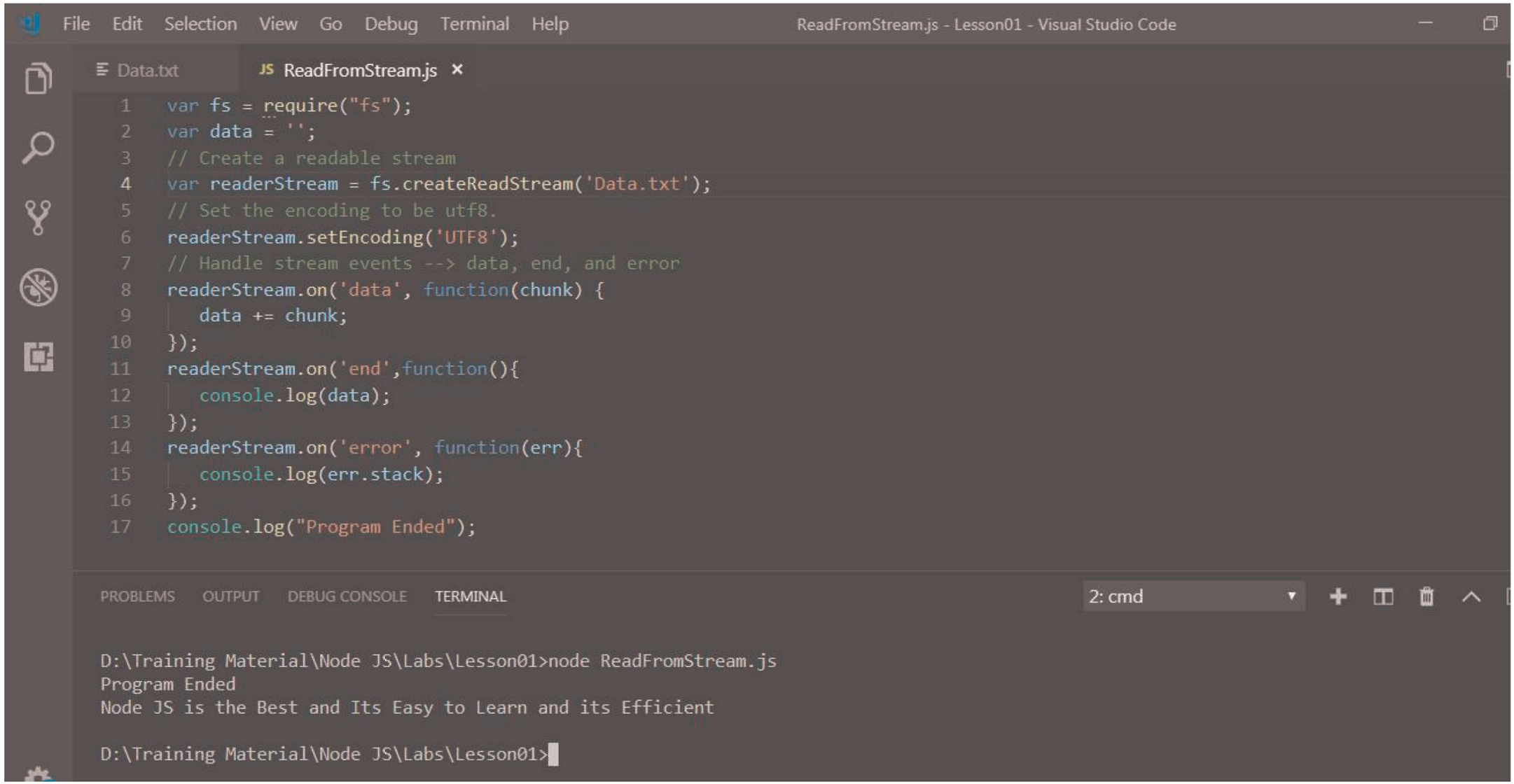

File Edit Selection View Go Debug Terminal Help
ReadFromBuffer.js - Lesson01 - Visual Studio Code
WriteToBuffers.js
ReadFromBuffer.js
1
2 buf = new Buffer(26);
3 for (var i = 0 ; i < 26 ; i++) {
4 buf[i] = i + 97;
5 }
6 console.log(buf.toString('ascii')); // outputs: abcdefghijklmnopqrstuvwxyz
7 console.log(buf.toString('ascii',0,5)); // outputs: abcde
8 console.log(buf.toString('utf8',0,5)); // outputs: abcde
9 console.log(buf.toString(undefined,0,5)); // encoding defaults to 'utf8', outputs abcde
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2: cmd
D:\Training Material\Node JS\Labs\Lesson01>node ReadFromBuffer.js
abcdefghijklmnopqrstuvwxyz
abcde
abcde
abcde
(node:6464) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer
.allocUnsafe(), or Buffer.from() methods instead.
D:\Training Material\Node JS\Labs\Lesson01>

Node.js Streams

Streams are the objects that facilitate you to read data from a source and write data to a destination. There are four types of streams in Node.js:

- **Readable:** This stream is used for read operations.
- **Writable:** This stream is used for write operations.
- **Duplex:** This stream can be used for both read and write operations.
- **Transform:** It is type of duplex stream where the output is computed according to input.

Node.js Reading from stream



Node.js Writing to stream