**Node Js Tutorial**

Node.js is a cross-platform runtime environment and library for running JavaScript applications outside the browser. It is used for creating server-side and networking web applications. It is open source and free to use. It can be downloaded from this link <https://nodejs.org/en/>

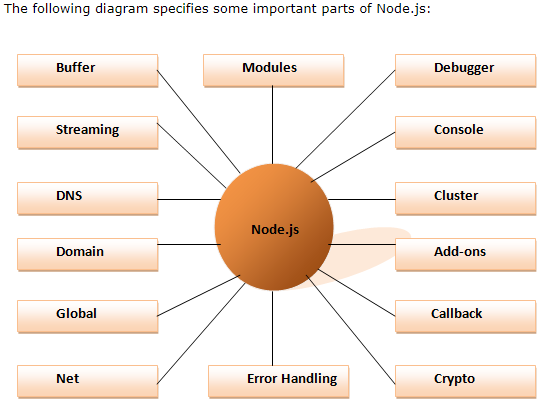
Many of the basic modules of Node.js are written in JavaScript. Node.js is mostly used to run real-time server applications.

The definition given by its official documentation is as follows:

?Node.js is a platform built on Chrome's JavaScript runtime for easily building fast and scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.?

Node.js also provides a rich library of various JavaScript modules to simplify the development of web applications.





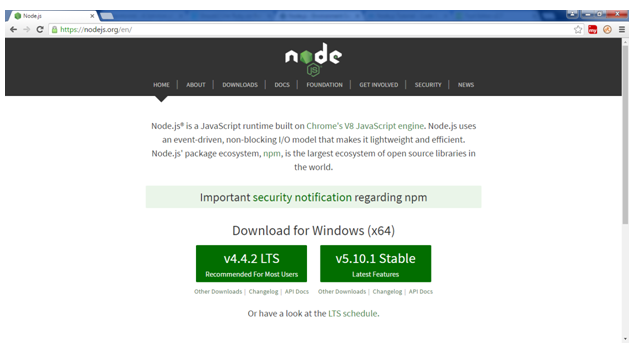
Features of Node.js

Following is a list of some important features of Node.js that makes it the first choice of software architects.

1. **Extremely fast:**Node.js is built on Google Chrome's V8 JavaScript Engine, so its library is very fast in code execution.
2. **I/O is Asynchronous and Event Driven:**All APIs of Node.js library are asynchronous i.e. non-blocking. So a Node.js based server never waits for an API to return data. The server moves to the next API after calling it and a notification mechanism of Events of Node.js helps the server to get a response from the previous API call. It is also a reason that it is very fast.
3. **Single threaded:**Node.js follows a single threaded model with event looping.
4. **Highly Scalable:**Node.js is highly scalable because event mechanism helps the server to respond in a non-blocking way.
5. **No buffering:**Node.js cuts down the overall processing time while uploading audio and video files. Node.js applications never buffer any data. These applications simply output the data in chunks.
6. **Open source:**Node.js has an open source community which has produced many excellent modules to add additional capabilities to Node.js applications.
7. **License:**Node.js is released under the MIT license.

**How to download Node.js:**

You can download the latest version of Node.js installable archive file from <https://nodejs.org/en/>

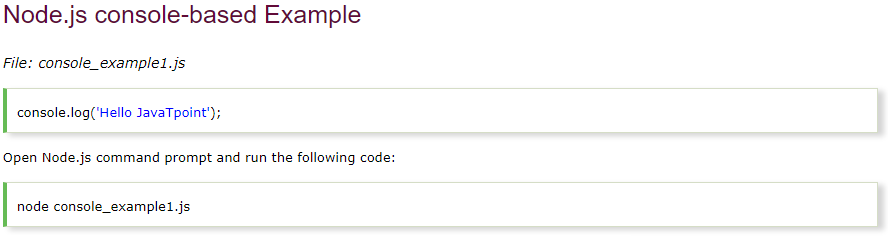


# Node.js First Example

There can be

1)console-based and

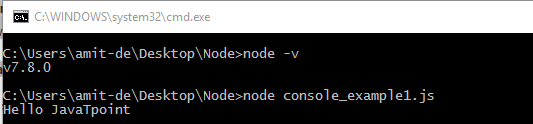
2)web-based node.js applications.





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

1. node console\_example1.js



## **Node.js web-based Example**

A node.js web application contains the following three parts:

1. **Import required modules:** The "require" directive is used to load a Node.js module.
2. **Create server:**You have to establish a server which will listen to client's request similar to Apache HTTP Server.
3. **Read request and return response:** Server created in the second step will read HTTP request made by client which can be a browser or console and return the response.

**How to create node.js web applications**

Follow these steps:

1. **Import required module:**The first step is to use ?require? directive to load http module and store returned HTTP instance into http variable. For example:



1. **Create server:**In the second step, you have to use created http instance and call http.createServer() method to create server instance and then bind it at port 8081 using listen method associated with server instance. Pass it a function with request and response parameters and write the sample implementation to return "Hello World". For example:

http.createServer(function (request, response) {

   // Send the HTTP header

   // HTTP Status: 200 : OK

   // Content Type: text/plain

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

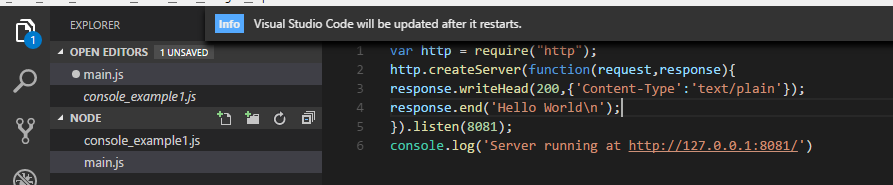
   // Send the response body as "Hello World"

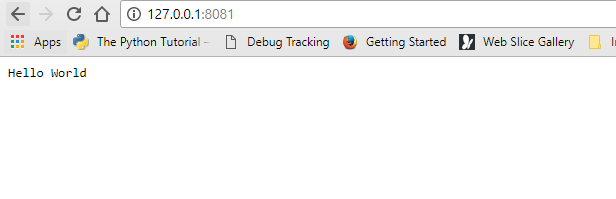
   response.end('Hello World\n');

}).listen(8081);

// Console will print the message

console.log('Server running at http://127.0.0.1:8081/');





Node.js Console

The Node.js console module provides a simple debugging console similar to JavaScript console mechanism provided by web browsers.

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

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

1-The console.log() function is used to display simple message on console.

*File: console\_example1.js*

console.log('Hello JavaTpoint');



we can also use format specifier in console.log() function.

*File: console\_example2.js*

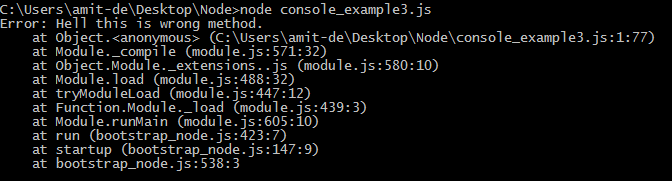
console.log('Hello %s', 'JavaTpoint');



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

*File: console\_example3.js*

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



# Node.js REPL

The term REPL stands for **Read Eval Print**and**Loop**. It specifies a computer environment like a window console or a Unix/Linux shell where you can enter the commands and the system responds with an output in an interactive mode.

## **REPL Environment**

The Node.js or node come bundled with REPL environment. Each part of the REPL environment has a specific work.

**Read:** It reads user's input; parse the input into JavaScript data-structure and stores in memory.

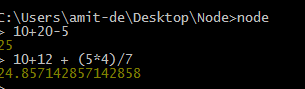
**Eval:**It takes and evaluates the data structure.

**Print:**It prints the result.

**Loop:** It loops the above command until user press ctrl-c twice.

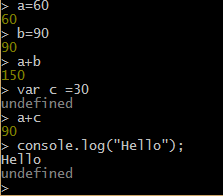
## **How to start REPL**

You can start REPL by simply running "node" on the command prompt. See this:



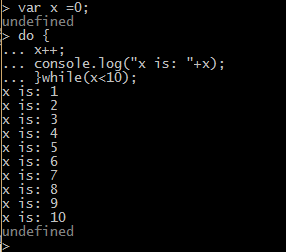
## **Using variable**

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().



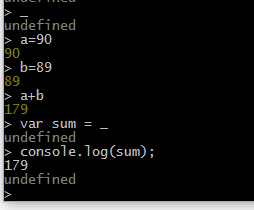
## **Node.js Multiline expressions**

Node REPL supports multiline expressions like JavaScript. See the following do-while loop example:



## **Node.js Underscore Variable**

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



## **Node.js REPL Commands**

|  |  |
| --- | --- |
| Commands | Description |
| ctrl + c | It is used to terminate the current command. |
| ctrl + c twice | It terminates the node repl. |
| ctrl + d | It terminates the node repl. |
| up/down keys | It is used to see command history and modify previous commands. |
| tab keys | It specifies the list of current command. |
| .help | It specifies the list of all commands. |
| .break | It is used to exit from multi-line expressions. |
| .clear | It is used to exit from multi-line expressions. |
| .save filename | It saves current node repl session to a file. |
| .load filename | It is used to load file content in current node repl session. |

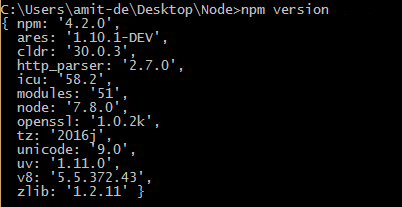
# Node.js Package Manager

Node Package Manager provides two main functionalities:

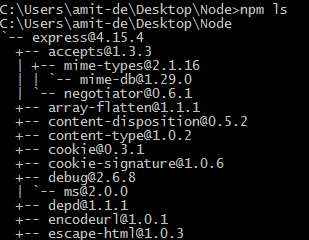
* It provides online repositories for node.js packages/modules which are searchable on search.nodejs.org
* It also provides command line utility to install Node.js packages, do version management and dependency management of Node.js packages.

The npm comes bundled with Node.js installables in versions after that v0.6.3. You can check the version by opening Node.js command prompt and typing the following command:

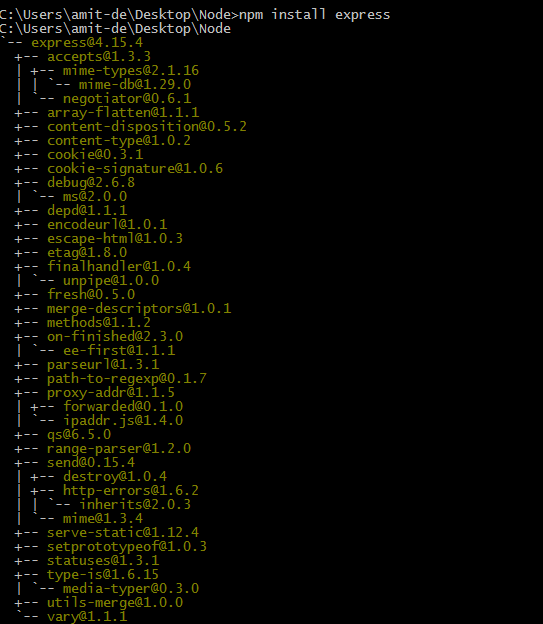
1. npm  version



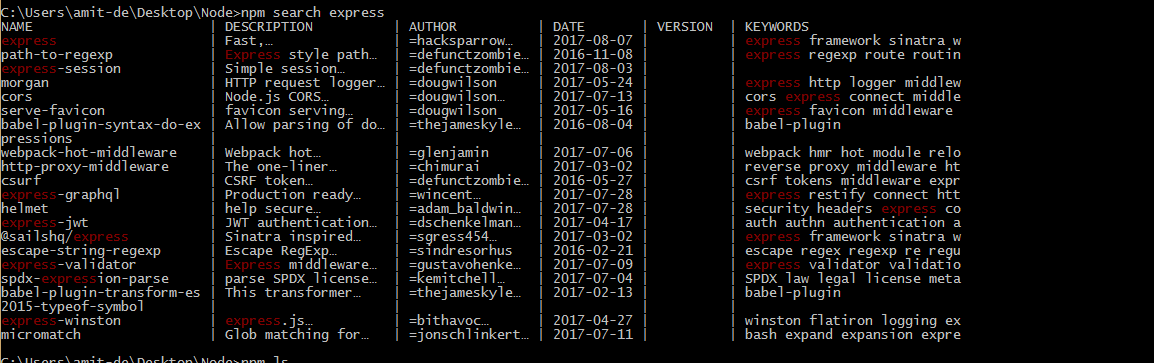
2- You can use npm ls command to list down all the locally installed modules.



1. 3- npm install express -g



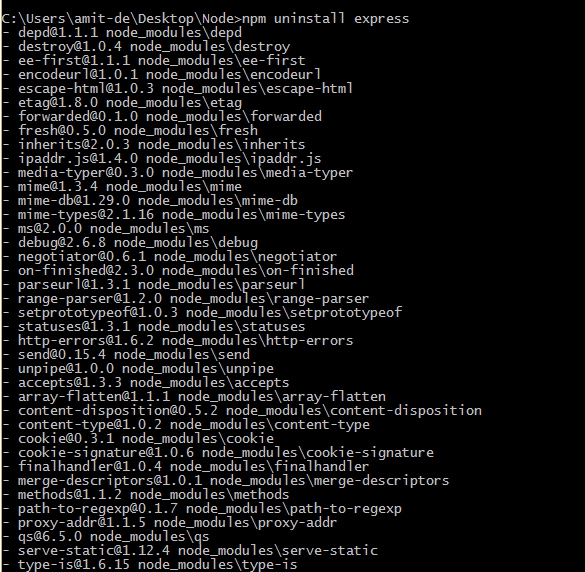
## 4-Searching of modules



## 5- **Uninstalling a Module**

To uninstall a Node.js module, use the following command:

1. npm uninstall express



|  |  |  |
| --- | --- | --- |
| **Index** | **Option** | **Description** |
| 1. | v, --version | It is used to print node's version. |
| 2. | -h, --help | It is used to print node command line options. |
| 3. | -e, --eval "script" | It evaluates the following argument as JavaScript. The modules which are predefined in the REPL can also be used in script. |
| 4. | -p, --print "script" | It is identical to -e but prints the result. |
| 5. | -c, --check | Syntax check the script without executing. |
| 6. | -i, --interactive | It opens the REPL even if stdin does not appear to be a terminal. |
| 7. | -r, --require module | It is used to preload the specified module at startup. It follows require()'s module resolution rules. Module may be either a path to a file, or a node module name. |
| 8. | --no-deprecation | Silence deprecation warnings. |
| 9. | --trace-deprecation | It is used to print stack traces for deprecations. |
| 10. | --throw-deprecation | It throws errors for deprecations. |
| 11. | --no-warnings | It silence all process warnings (including deprecations). |
| 12. | --trace-warnings | It prints stack traces for process warnings (including deprecations). |
| 13. | --trace-sync-io | It prints a stack trace whenever synchronous i/o is detected after the first turn of the event loop. |
| 14. | --zero-fill-buffers | Automatically zero-fills all newly allocated buffer and slowbuffer instances. |
| 15. | --track-heap-objects | It tracks heap object allocations for heap snapshots. |
| 16. | --prof-process | It processes V8 profiler output generated using the v8 option --prof. |
| 17. | --V8-options | It prints V8 command line options. |
| 18. | --tls-cipher-list=list | It specifies an alternative default tls cipher list. (requires node.js to be built with crypto support. (default)) |
| 19. | --enable-fips | It enables fips-compliant crypto at startup. (requires node.js to be built with ./configure --openssl-fips) |
| 20. | --force-fips | It forces fips-compliant crypto on startup. (cannot be disabled from script code.) (same requirements as --enable-fips) |
| 21. | --icu-data-dir=file | It specifies ICU data load path. (Overrides node\_icu\_data) |

Node.js Command Line Options

There is a wide variety of command line options in Node.js. These options provide multiple ways to execute scripts and other helpful run-time options.

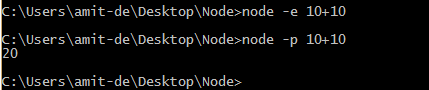
Let's see the list of Node.js command line options:

## **To evaluate an argument (but not print result):**

Use command node -e, --eval "script"

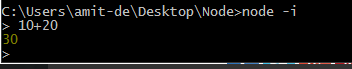
## **To evaluate an argument and print result also:**

Use command node -p "script"



## **To open REPL even if stdin doesn't appear:**

Use command node -i, or node --interactive



# Node.js Global 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.

A list of Node.js global objects are given below:

* \_\_dirname
* \_\_filename
* Console
* Process
* Buffer
* setImmediate(callback[, arg][, ...])
* setInterval(callback, delay[, arg][, ...])
* setTimeout(callback, delay[, arg][, ...])
* clearImmediate(immediateObject)
* clearInterval(intervalObject)
* clearTimeout(timeoutObject)

## **Node.js \_\_dirname**

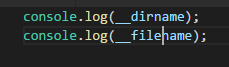
It is a string. It specifies the name of the directory that currently contains the code.

## **Node.js \_\_filename**

It specifies the filename of the code being executed. This is the resolved absolute path of this code file. The value inside a module is the path to that module file.

*File: global-example2.js*

1. console.log(\_\_filename);





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. |
| 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. |

# Node.js Timer

Node.js Timer functions are global functions. You don't need to use require() function in order to use timer functions. Let's see the list of timer functions.

**Set timer functions:**

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

**Clear timer functions:**

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

## **Node.js Timer setInterval() Example**

This example will set a time interval of 1000 millisecond and the specified comment will be displayed after every 1000 millisecond until you terminate.

****

File: timer1.js

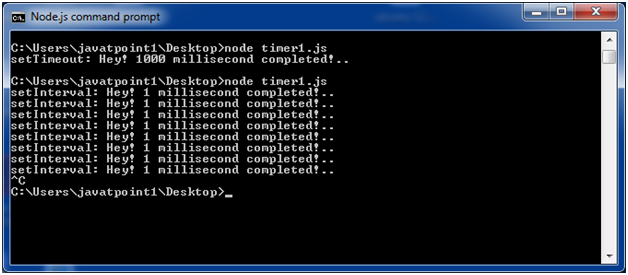
1. setInterval(function() {
2. console.log("setInterval: Hey! 1 millisecond completed!..");
3. }, 1000);



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

1. node timer1.js





File: timer5.js

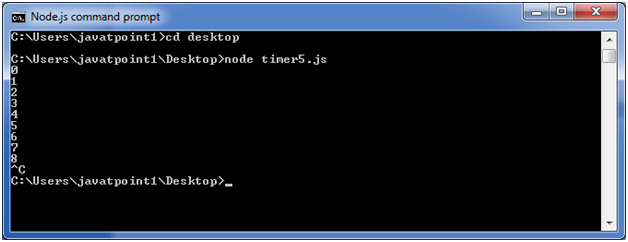
1. var i =0;
2. console.log(i);
3. setInterval(function(){
4. i++;
5. console.log(i);
6. }, 1000);



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

1. node timer5.js





## **Node.js Timer setTimeout() Example**

File: timer1.js

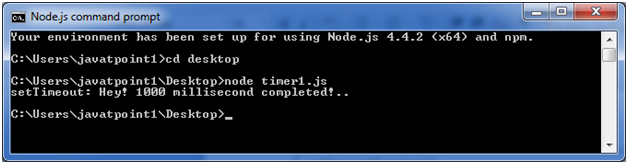
1. setTimeout(function() {
2. console.log("setTimeout: Hey! 1000 millisecond completed!..");
3. }, 1000);



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

1. node timer1.js





This example shows time out after every 1000 millisecond without setting a time interval. This example uses the recursion property of a function.

File: timer2.js

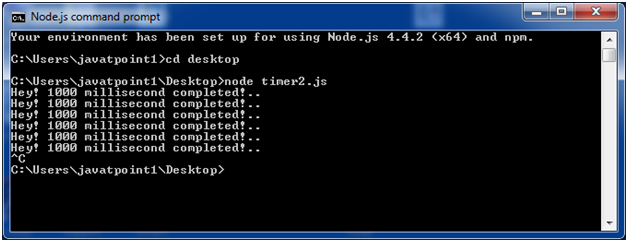
1. var recursive = function () {
2. console.log("Hey! 1000 millisecond completed!..");
3. setTimeout(recursive,1000);
4. }
5. recursive();



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

1. node timer2.js





## **Node.js setInterval(), setTimeout() and clearTimeout()**

Let's see an example to use clearTimeout() function.

File: timer3.js

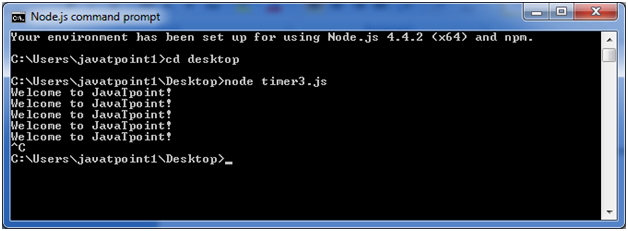
1. function welcome () {
2. console.log("Welcome to JavaTpoint!");
3. }
4. var id1 = setTimeout(welcome,1000);
5. var id2 = setInterval(welcome,1000);
6. clearTimeout(id1);
7. //clearInterval(id2);



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

1. node timer3.js





You can see that the above example is recursive in nature. It will terminate after one step if you use ClearInterval.

## **Node.js setInterval(), setTimeout() and clearInterval()**

Let's see an example to use clearInterval() function.

File: timer3.js

1. function welcome () {
2. console.log("Welcome to JavaTpoint!");
3. }
4. var id1 = setTimeout(welcome,1000);
5. var id2 = setInterval(welcome,1000);
6. //clearTimeout(id1);
7. clearInterval(id2);



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

1. node timer3.js





# Node.js Errors

The Node.js applications generally face four types of errors:

* **Standard JavaScript errors** i.e. <EvalError>, <SyntaxError>, <RangeError>, <ReferenceError>, <TypeError>, <URIError> etc.
* **System errors**
* **User-specified errors**

**Assertion errors**

## **Node.js Errors Example 1**

Let's take an example to deploy standard JavaScript error - ReferenceError.

File: error\_example1.js

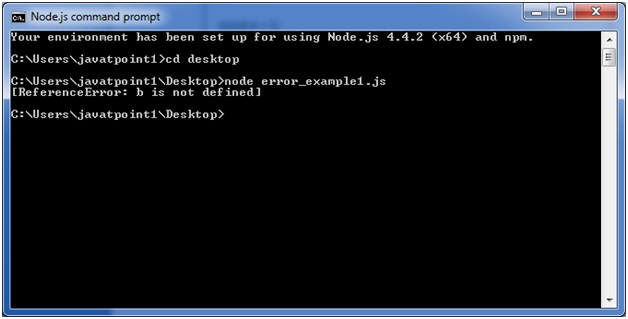
1. // Throws with a ReferenceError because b is undefined
2. try {
3. const a = 1;
4. const c = a + b;
5. } catch (err) {
6. console.log(err);
7. }



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

1. node error\_example1.js





## **Node.js Errors Example 2**

File: timer2.js

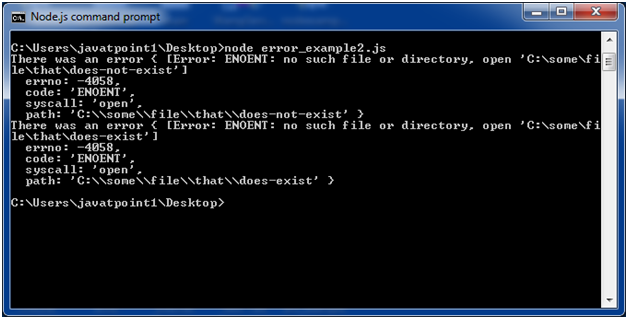
1. const fs = require('fs');
2. function nodeStyleCallback(err, data) {
3. if (err) {
4. console.error('There was an error', err);
5. return;
6. }
7. console.log(data);
8. }
9. fs.readFile('/some/file/that/does-not-exist', nodeStyleCallback);
10. fs.readFile('/some/file/that/does-exist', nodeStyleCallback);



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

1. node error\_example2.js





****