

Node.js console.warn()

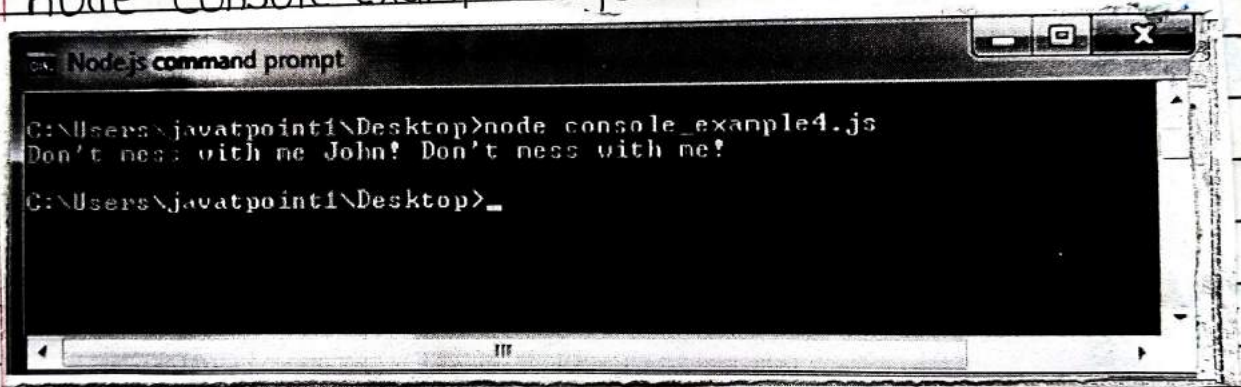
The `console.warn()` function is used to display warning message on console.

File: `console_example4.js`

```
const name = 'John';  
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_example4.js
```



A screenshot of a Windows command prompt window titled "Node.js command prompt". The window shows the command `node console_example4.js` being executed at the path `C:\Users\javatpoint\Desktop>`. The output displayed is `Don't mess with me John! Don't mess with me!`. The prompt then returns to `C:\Users\javatpoint\Desktop>`.

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; parses 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 presses ctrl-c twice.

Node.js Simple expressions

After starting REPL node command prompt put any mathematical expression:

Example: $> 10 + 20 - 5$

25

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()`.

Example: $> a = 50$

50

$> \text{var } b = 50$

undefined

$> a + b$

100

Node.js Multiline expressions

Node REPL supports multiline expressions like Java Script.

Example

```
Var x = 0
```

```
Undefined
```

```
> do {
```

```
... x ++;
```

```
... console.log("x:" + x);
```

```
... } while (x < 10);
```

Node.js Underscore Variable

You can also use underscore to get the last result.

Example

```
C:\Users\javatpoint\1\Desktop> node
```

```
> var a = 50
```

```
Undefined
```

```
> var b = 50
```

```
Undefined
```

```
> a + b
```

```
100
```

Node.js REPL Commands

1. Ctrl + C :

It is used to terminate the current command.

2. Ctrl + C twice :

It terminates the node repl.

3. Ctrl + d :

It terminates the node repl.

4. up/down keys:

It is used to see Command history and modify previous commands.

5. tab keys:

It specifies the list of current command.

6. help:

It specifies the list of all commands.

7. break:

It is used to exit from multi-line expressions.

8. Clear:

It is used to exit from multi-line expressions.

9. save filename:

It saves current node repl session to a file.

10. load filename:

It is used to load file content in current node repl session.

Node.js Exit REPL

Use Ctrl+C Command twice to come out of Node.js REPL.

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.npmjs.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 version after that v0.6.3. You can check the version by opening Node.js Command prompt and typing the following command:

```
npm version
```

Installing modules using npm

```
npm install <module Name>
```

Uninstalling a Module

```
npm uninstall express
```

Searching a module

```
npm search express
```

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.

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 set 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 silences all process warnings.
12. --trace-warnings
It prints stack traces for process warnings.
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-chiper-list=list
It specifies an alternative default tls chiper 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)

21. --icu-data-dir=file

It specifies ICU data load path.

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.

Node.js __dirname

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

```
console.log(__dirname)
```

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

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


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

node global-example2.js

Node.js Console

click here to get details of Console class

<http://www.javatpoint.com/nodejs-console>

Node.js Buffer

Click here to get details of Buffer class.

<http://www.javatpoint.com/nodejs-buffers>

Node.js Timer Functions

Click here to get details of Timer functions.

<http://www.javatpoint.com/nodejs-timer>

Node.js OS

Node.js OS provides some basic operating-system related utility functions.

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 method 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 return 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 the 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.

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

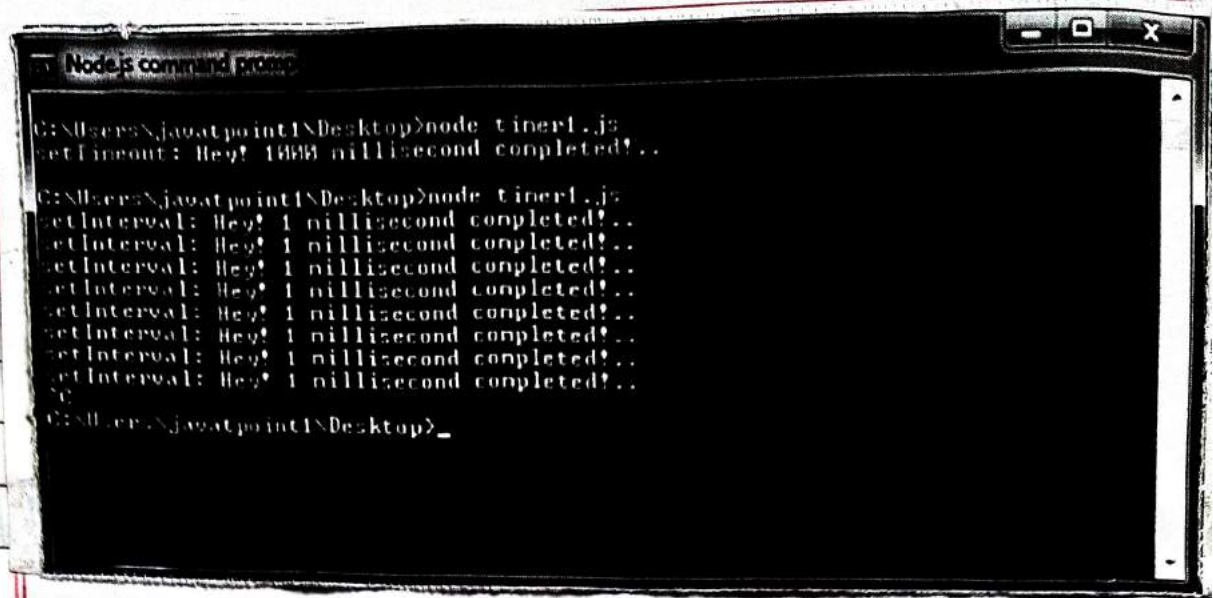
This example will set a time interval of 1000 milliseconds and the specified comment will be displayed after every 1000 milliseconds until you terminate.

File: timer 1.js

```
setInterval(function(){  
  console.log('SetInterval: Hey! 1 millisecond completed!..');  
}, 1000);
```

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

node timer 1.js



```
Node.js command prompt
C:\Users\javatpoint\Desktop>node timer1.js
setInterval: Hey! 1888 millisecond completed!...

C:\Users\javatpoint\Desktop>node timer1.js
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
setInterval: Hey! 1 millisecond completed!...
C:\Users\javatpoint\Desktop>_
```

Node.js Errors

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

- Standard JavaScript errors i.e.
<EvalError>, <Syntax Error>, <Range Error>, <Reference Error>, <Type Error>, <URI ERROR> etc.
- System errors
- Use-specified errors
- Assertion errors

Node.js Errors Example

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

File error_example1.js

// Throws With a ReferenceError because `b` is undefined

```
try {
```

```
  const a = 1;
```

```
  const c = a + b;
```

```
} catch (err) {
```

```
  console.log(err);
```

```
}
```

Open Node.js Command prompt and run the

following code:

node error_example 1.js



Node.js DNS

The Node.js DNS module contains methods to get information of given hostname. Lets see the list of commonly used DNS functions:

- `dns.getServers()`
- `dns.setServers(servers)`
- `dns.lookup(hostname[, options], callback)`
- `dns.lookupService(address, port, callback)`
- `dns.resolve(hostname[, rrtype], callback)`
- `dns.resolve4(hostname, callback)`
- `dns.resolve6(hostname, callback)`
- `dns.resolveCname(hostname, callback)`
- `dns.resolveMx(hostname, callback)`
- `dns.resolveNs(hostname, callback)`
- `dns.resolveSoa(hostname, callback)`
- `dns.resolveSrv(hostname, callback)`
- `dns.resolvePtr(hostname, callback)`
- `dns.resolveTxt(hostname, callback)`
- `dns.reverse(ip, callback)`

Node.js DNS Example

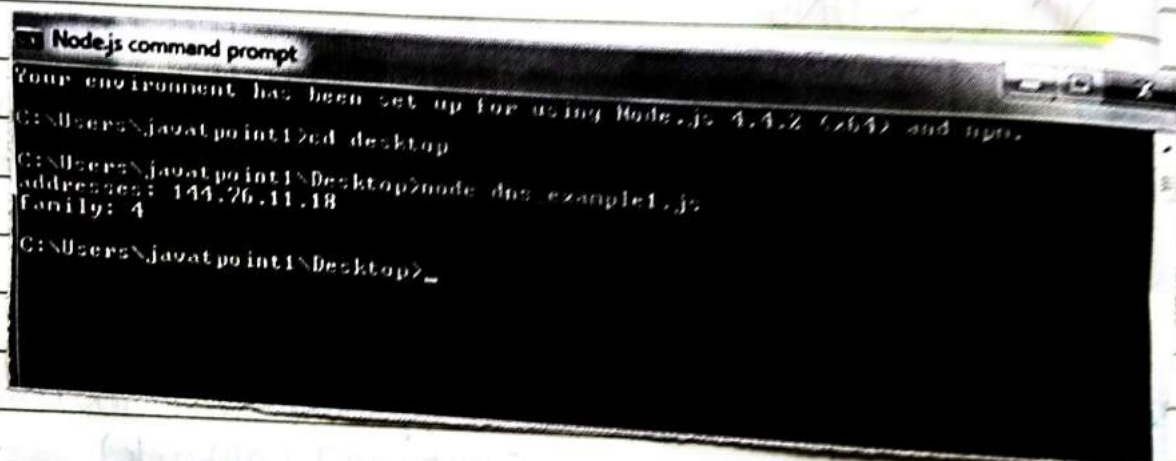
Lets see the example of `dns.lookup()` function.

File: dns_example1.js

```
Const dns = require('dns');  
dns.lookup('www.javatpoint.com', (err, addresses,  
          family) => {  
    Console.log('addresses:', addresses);  
    Console.log('family:', family);  
});
```

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

node dns_example1.js



Node.js Net

Node.js provides the ability to perform socket programming. We can create chat application or communicate client and server applications using socket programming in Node.js. The Node.js net module contains functions for creating both servers and clients.

Node.js Net Example

In this example, we are using two command prompts:

- Node.js command prompt for server.
- Window's default command prompt for client.