Node.js REPL

Mohamed Mukthar Ahmed

1

Node.js REPL

Outline

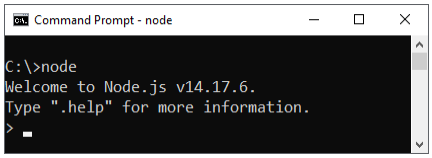
- What is REPL?
- Start REPL Session
- Autocomplete
- REPL Special Commands
- Special _ Variable
- Run REPL from JavaScript File
- Running Node.js Program
- Exit Node.js Program
- Command Line Arguments
- Program Output
  - Format Specifiers
  - Count and Reset Count
- Print Stack Trace
- Calculate Time Spent
- Print to STDOUT and STDERR

Mohamed Mukthar Ahmed

2

What is REPL?

- If we run the node command without any script to execute or without any arguments, we start a REPL session:
  - **REPL** stands for **R**ead **E**valuate **P**rint **L**oop, and it is a programming language environment (basically a console window).
  - REPL takes single expression as user input and returns the result back to the console after execution.
  - The REPL session provides a convenient way to quickly test simple JavaScript code.

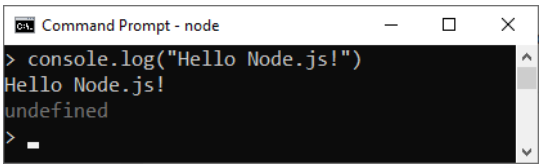


Mohamed Mukthar Ahmed

3

Node.js REPL

- **REPL** is waiting for us to enter some JavaScript code.



- Node read the line of code, evaluated it, printed the result, and then went back to waiting for more lines of code.
- Node will loop through these three steps for every piece of code we execute in the **REPL** until we exit the session.
- That is where the **REPL** got its name.

Mohamed Mukthar Ahmed

4

Autocomplete

- **REPL** is waiting for us to enter some JavaScript code.

```
> var x = "Hello, World!"
undefined
> x
"Hello, World!"
> .exit
```

- You can also use the **TAB** key to **autocomplete** some commands. When multiple autocomplete options are available, hit **TAB** again to cycle through them.
- You can press the **UP** and **DOWN arrow** keys to scroll through your command history and modify previous commands

Mohamed Mukthar Ahmed

5

REPL Special Commands

- **Special commands** support by REPL instance

```
> .help
.break    Sometimes you get stuck, this gets you out
.clear    Alias for .break
.editor   Enter editor mode
.exit     Exit the REPL
.help     Print this help message
.load     Load JS from a file into the REPL session
.save     Save all evaluated commands in this REPL session to a file

Press Ctrl+C to abort current expression, Ctrl+D to exit the REPL
>
```

```
> .editor
// Entering editor mode (Ctrl+D to finish, Ctrl+C to cancel)
function welcome(name) {
  return `Hello ${name}!`;
}

welcome('Node.js User')

// Ctrl+D

'Hello Node.js User!'
>
```

```
> .save C:\temp\ex_01.js
Session saved to: C:\temp\ex_01.js
>
```

Mohamed Mukthar Ahmed

6

Special _ Variable

- Helps to reuse the previous return value.

```
> 1+1
2
> _+1
3
```

```
> x = 10
10
> var y = 5
> x
10
> y
5
```

- When the **var** keyword is used, the value of the expression is stored, but **NOT** returned.
- When a bare identifier is used, the value is also returned, as well as stored..

```
> [11, 22, 33].forEach( num => {
...   console.log(num)
... })
11
22
33
undefined
>
```

Mohamed Mukthar Ahmed

7

Run REPL from JavaScript file

- We can import REPL in a JavaScript file using **repl** module.

```
// Using repl in JavaScript file
const local = require('repl');

local.start('$ ');  // Starting a REPL session

```

- Save it to a file by name ‘repl.js’

Mohamed Mukthar Ahmed

8

Running Node.js Program

- The usual way to run a Node.js program is to run the globally available **node** command and pass the name of the file you want to execute.
- If your main Node.js application file is **app.js**, you can call it by typing:

```bash
node app.js
```

- Running the Node.js program saved in a file by name **repl.js**

```
Command Prompt - node repl.js

D:\NodeJS-Demos>node repl.js
$
```

Mohamed Mukthar Ahmed

9

Exit a Node.js Program

- The **process** core module provides a handy method that allows you to programmatically exit from a Node.js program: **process.exit().**
  - When Node.js runs this line, the process is immediately forced to terminate.
- This means that any callback that's pending, any network request still being sent, any filesystem access, or processes writing to **stdout** or **stderr** - all is going to be ungracefully terminated right away.
- You can pass an integer that signals the operating system the **exit status** (exit code)

Mohamed Mukthar Ahmed

10

Command Line Arguments

- You can pass any number of arguments when invoking a Node.js application
- Arguments can be standalone or have a key and a value.
- The way you retrieve it is using the **process** object **argv** property. It's an array that contains all the command line invocation arguments.
  - The first element is the full path of the node command.
  - The second element is the full path of the file being executed.
  - All the additional arguments are present from the third position going forward.

Mohamed Mukthar Ahmed

11

Command Line Arguments

```
// Command Line Arguments
console.log('Example: Command Line Arguments...');

process.argv.forEach( (value, index) => {
    console.log(`${index}: ${value}`);
});
```

```
D:\NodeJS-Demos>node cmd-line-args.js
Example: Command Line Arguments...
0: C:\ProgramData\nodejs\node.exe
1: D:\NodeJS-Demos\cmd-line-args.js

D:\NodeJS-Demos>node cmd-line-args.js Mukthar 45
Example: Command Line Arguments...
0: C:\ProgramData\nodejs\node.exe
1: D:\NodeJS-Demos\cmd-line-args.js
2: Mukthar
3: 45
```

Mohamed Mukthar Ahmed

12

Node.js Program Output

- Node.js provides a **console** module for this purpose.
- The most basic and most used method is **console.log()**, which prints the string you pass to it to the console.
  - If you pass an object, it will render it as a string.
- We can also format pretty phrases by passing variables and a format specifier.
  - **%s** format a variable as a string
  - **%d** format a variable as a number
  - **%i** format a variable as its integer part only
  - **%o** format a variable as an object

Mohamed Mukthar Ahmed

13

Node.js Program Output

```
// More on console methods
const num = Number(12);

console.log('Example: More on console methods...');
console.log('Using format specifiers:');
console.log('My %s has %d ears', 'cat', 2);
console.log('The number is: %o', num);
```

```
Example: More on console methods...
Using format specifiers:
My cat has 2 ears
The number is: 12
```

Mohamed Mukthar Ahmed

14

Node.js Program Output

- Counting elements
  - The **console.count()** is a handy method.

```
// Understanding console.count() method
const x = 1
const y = 2
const z = 3
console.count(
  'The value of x is ' + x +
  ' and has been checked .. how many times?'
)
console.count(
  'The value of x is ' + x +
  ' and has been checked .. how many times?'
)
console.count(
  'The value of y is ' + y +
  ' and has been checked .. how many times?'
)
```

Mohamed Mukthar Ahmed

15

Node.js Program Output

- Counting elements
  - The **console.count()** is a handy method.

```
// Understanding console.count() method
const oranges = ['orange', 'orange'];
const apples = ['just one apple'];
oranges.forEach(fruit => {
  console.count(fruit);
});
apples.forEach(fruit => {
  console.count(fruit);
});
```

- Reset counting
  - The **console.countReset()** method resets counter used with console.count()

Mohamed Mukthar Ahmed

16

Print Stack Trace

- There might be cases where it's useful to print the call stack trace of a function, maybe to answer the question *how did you reach that part of the code*?
- You can do so using **console.trace()**

```
const function2 = () => console.trace();
const function1 = () => function2();
function1();
```

Mohamed Mukthar Ahmed

17

Calculate Time Spent

- You can easily calculate how much time a function takes to run, using **time()** and **timeEnd()** methods

```
// Calculate time spent
const doSomething = () => console.log('Testing...');
const measureDoingSomething = () => {
  console.time('doSomething()');
  // do something, and measure the time it takes
  doSomething();
  console.timeEnd('doSomething()');
};
measureDoingSomething();
```

- Modify the **doSomething()** function which does slightly more (use an iterative construct) and record your observation.

Mohamed Mukthar Ahmed

18

Print to STDOUT and STDERR

- The **console.log()** prints on the **STDOUT** stream
- On the other hand **console.error()** prints to the **STDERR** stream
- Will it make any difference?
  - YES. When we perform **REDIRECTION.**
- You can color the output of your text in the console by using **escape sequences**.
  - An escape sequence is a set of characters that identifies a color.

```
// Using Esc Seq for coloring
console.log('\x1b[33m %s \x1b[0m', 'Hello!');
```

Mohamed Mukthar Ahmed