CoffeeScript

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

CoffeeScript is a lightweight language which transcompiles into JavaScript. It provides better syntax avoiding the quirky parts of JavaScript, still retaining the flexibility and beauty of the language.

Audience

This tutorial has been prepared for beginners to help them understand the basic functionality of CoffeeScript to build dynamic webpages and web applications.

Prerequisites

For this tutorial, it is assumed that the readers have a prior knowledge of HTML coding and JavaScript. It would help if the reader has some prior exposure to object-oriented programming concepts and a general idea on creating online applications.

Execute CoffeeScript Online

For most of the examples given in this tutorial, you will find **Try it** option, so just make use of this option to transcompile your CoffeeScript programs to JavaScript programs on the spot and enjoy your learning.

Try the following example using the **Try it** option available at the top right corner of the below sample code box –

```
console.log "Hello Welcome to Tutorials point"
```

Copyright & Disclaimer

Table of Contents

About the Tutorial i

Audience i

Prerequisites i

Execute CoffeeScript Online i

Copyright & Disclaimer i

Table of Contents ii

1. COFFEESCRIPT – OVERVIEW 1

What is CoffeeScript ? 1

History of CoffeeScript 1

Limitations of CoffeeScript 2

TutorialsPoint's CoffeeScript IDE 2

2. COFFEESCRIPT – ENVIRONMENT 4

Node.js 4

Installing Node.js 4

Installing CoffeeScript 8

3. COFFEESCRIPT – COMMAND-LINE UTILITY 9

Compiling the CoffeeScript Code 9

Executing the CoffeeScript code 11

Watch and Compile 11

Setting the Output Directory 12

Print the Compiled JavaScript 12

The REPL (Read Evaluate Print Loop) 13

Running CoffeeScript through Browser 14

4. COFFEESCRIPT – SYNTAX 15

CoffeeScript Statements 15

CoffeeScript Variables (No var Keyword) 15

No Parentheses 15

No Curly Braces 16

CoffeeScript Comments 16

CoffeeScript Reserved keywords 17

5. COFFEESCRIPT – DATA TYPES 18

CoffeeScript Data Types 18

6. COFFEESCRIPT – VARIABLES 19

CoffeeScript Variables 19

Variable Scope 19

The Problem with Variables in JavaScript 20

Variable Scope in CoffeeScript 20

CoffeeScript Variable Names (Literals) 21

7. COFFEESCRIPT – OPERATORS 22

CoffeeScript Aliases 22

Arithmetic Operators 22

Comparison Operators 25

Logical Operators 30

Bitwise Operators 34

Assignment Operators 37

Equality Operator in CoffeeScript 40

The existential Operator 41

Chained Comparisons 42

CoffeeScript Aliases 42

8. COFFEESCRIPT – CONDITIONALS 46

if Statement 47

if...else Statement 49

unless Statement 50

unless…else Statement 52

switch Statement 54

Switch Statement in CoffeeScript 54

Multiple values for when clause 57

The then Keyword in CoffeeScript 58

if…then Statement 59

if…then…else Statement 60

unless…then Statement 61

unless-then…else Statement 62

postfix if and postfix unless Statements 63

9. COFFEESCRIPT – LOOPS 65

The while loop in CoffeeScript 65

Variants of while 67

The until Variant of while 67

The loop Variant of while 69

10. COFFEESCRIPT – COMPREHENSIONS 72

for…in Comprehensions 72

for…of Comprehensions 73

list Comprehensions 74

Index of comprehensions 77

Postfix form of comprehensions 78

Postfix for..in comprehension 78

Assigning to a variable 80

The by keyword 81

11. COFFEESCRIPT – FUNCTIONS 83

Functions in CoffeeScript 83

Defining a Function 83

Multi-line Functions 84

Functions with Arguments 85

Invoking a Function 85

Invoking Functions with Arguments 86

Default Arguments 87

12. COFFEESCRIPT – STRINGS 90

String Concatenation 90

String Interpolation 91

JavaScript String Object 92

charAt method 93

charCodeAt Method 95

concat() Method 96

indexOf() Method 98

lastIndexOf() Method 99

localeCompare() Method 100

match() Method 104

search() Method 105

slice() Method 107

split() Method 108

substr() Method 109

toLocaleLowerCase() method 111

toLocaleUpperCase() method 112

toLowerCase() method 113

toUpperCase() method 114

13. COFFEESCRIPT – ARRAYS 116

New line instead of comma 116

Comprehensions over arrays 117

14. COFFEESCRIPT – OBJECTS 119

Indentations instead of curly braces 120

Nested objects 120

Comprehensions over objects 121

Arrays of Objects 122

Reserved Keywords 124

15. COFFEESCRIPT – RANGES 126

Excluding the end Value 127

Using Ranges with Variables 128

Ranges with Arrays 129

Ranges with Strings 131

Comprehensions over Ranges 132

16. COFFEESCRIPT – SPLAT 134

Calling Functions using Splats 136

Splats with a Tailing Argument 139

Comprehensions with Splats 141

17. COFFEESCRIPT – DATE 144

getDate() Method 146

getDay() Method 147

getFullYear() Method 148

getHours() Method 149

getMilliseconds() Method 150

getMinutes() Method 151

getMonth() Method 152

getUTCSeconds() Method 153

getTime() Method 154

getTimezoneOffset() Method 155
getUTCDate() Method 156
getUTCDay() method 157
getUTCFullYear() Method 158
getUTCHours() Method 159
getUTCMilliseconds() Method 160
getUTCMinutes() Method 161
getUTCMonth() Method 162
getUTCSeconds() Method 163
getYear() Method 164
setDate() Method 165
setFullYear() Method 166
setHours() Method 167
setMilliseconds() Method 168
setMinutes() Method 169
setMonth() Method 170
setSeconds() Method 171
setTime() Method 173
setUTCDate() Method 174
setUTCFullYear() Method 175
setUTCHours() Method 176
setUTCMilliseconds() Method 178
setUTCMinutes() Method 179
setUTCMonth() Method 180
setUTCSeconds() Method 181
setYear() Method 183
toDateString() Method 184
toLocaleDateString() Method 185

toLocaleString() Method **186**

toLocaleTimeString() Method **187**

toTimeString() Method **188**

toTimeString Method **189**

18. COFFEESCRIPT – MATH 191

Math Methods **193**

abs() Method **194**

acos() Method **196**

asin() Method **197**

atan() Method **198**

atan2() Method **200**

ceil() Method **201**

cos() Method **203**

exp() Method **204**

floor() Method **206**

log() Method **207**

max() Method **209**

min() Method **210**

pow() Method **212**

random() Method **213**

round() Method **215**

sin() Method **216**

sqrt() Method **218**

tan() Method **219**

19. COFFEESCRIPT – EXCEPTION HANDLING 222

Exceptions in CoffeeScript **222**

The finally block **224**

The throw Statement 226

20. COFFEESCRIPT – REGULAR EXPRESSIONS 228

Regular Expressions in CoffeeScript 228

heregex 229

21. COFFEESCRIPT – CLASSES AND INHERITANCE 231

Classes in CoffeeScript 231

Instantiating a class 231

Defining a Constructor 232

Static Properties 235

Inheritance 236

The Fat Arrow (=>) 238

22. COFFEESCRIPT – AJAX 239

CoffeeScript with Ajax 240

23. COFFEESCRIPT – JQUERY 242

Using CoffeeScript with jQuery 242

What is Callback? 244

24. COFFEESCRIPT – MANGODB 247

Installation 247

Connecting to MangoDB 247

Creating a Collection 248

Inserting Documents 249

Reading Documents 250

Updating Documents 253

Deleting Documents 254

25. COFFEESCRIPT – SQLITE 256

Installation 256

Connecting to Database **256**

Creating a Table **257**

Inserting / Creating Data **257**

Reading / Retrieving Data **258**

Updating Data **259**

Deleting Data **260**

1. COFFEESCRIPT – OVERVIEW

At present, JavaScript is the fastest mainstream dynamic language available, and it is known as the *lingua franca* of the web. It is developed by Brendan Eich in the year of 1995 in 10 days.

Because of its effective features, JavaScript became popular and went global quickly. It was there in lab for a very less time, which was not enough to polish the language. May be for this reason, inspite of its good parts, JavaScript has a bunch of design errors and it bagged a bad reputation of being a quirky language.

What is CoffeeScript ?

CoffeeScript is a lightweight language based on Ruby and Python which **transcompiles** (compiles from one source language to another) into JavaScript. It provides better syntax avoiding the quirky parts of JavaScript, still retaining the flexibility and beauty of the language.

Advantages of CoffeeScript

Following are the advantages of CoffeeScript –

- **Easily understandable** – CoffeeScript is a shorthand form of JavaScript, its syntax is pretty simple compared to JavaScript. Using CoffeeScript, we can write clean, clear, and easily understandable codes.
- **Write less do more** – For a huge code in JavaScript, we need comparatively very less number of lines of CoffeeScript.
- **Reliable** – CoffeeScript is a safe and reliable programming language to write dynamic programs.
- **Readable and maintainable** – CoffeeScript provides aliases for most of the operators which makes the code readable. It is also easy to maintain the programs written in CoffeeScript.
- **Class-based inheritance** – JavaScript does not have classes. Instead of them, it provides powerful but confusing prototypes. Unlike JavaScript, we can create classes and inherit them in CoffeeScript. In addition to this, it also provides instance and static properties as well as **mixins**. It uses JavaScript's native prototype to create classes.
- **No var keyword** – There is no need to use the **var** keyword to create a variable in CoffeeScript, thus we can avoid the accidental or unwanted scope deceleration.
- **Avoids problematic symbols** – There is no need to use the problematic semicolons and parenthesis in CoffeeScript. Instead of curly braces, we can use whitespaces to differentiate the block codes like functions, loops, etc.

- **Extensive library support** – In CoffeeScript, we can use the libraries of JavaScript and vice versa. Therefore, we have access to a rich set of libraries while working with CoffeeScript.

History of CoffeeScript

- CoffeeScript is developed by Jeremy Ashkenas. It was first committed in Git On December 13, 2009.
- Originally the compiler of the CoffeeScript was written in Ruby language.
- In March 2010, the CoffeeScript compiler was replaced; this time instead of Ruby, they used CoffeeScript itself.
- And in the same year, CoffeeScript 1.0 was released and at the time of release, it was one of the most wanted projects of the Git hub.

Limitations of CoffeeScript

Sensitive to whitespaces – CoffeeScript is very sensitive to whitespaces, so programmers need to be very careful while providing indentations. If we do not maintain proper indentation, the entire code may go wrong.

TutorialsPoint's CoffeeScript IDE

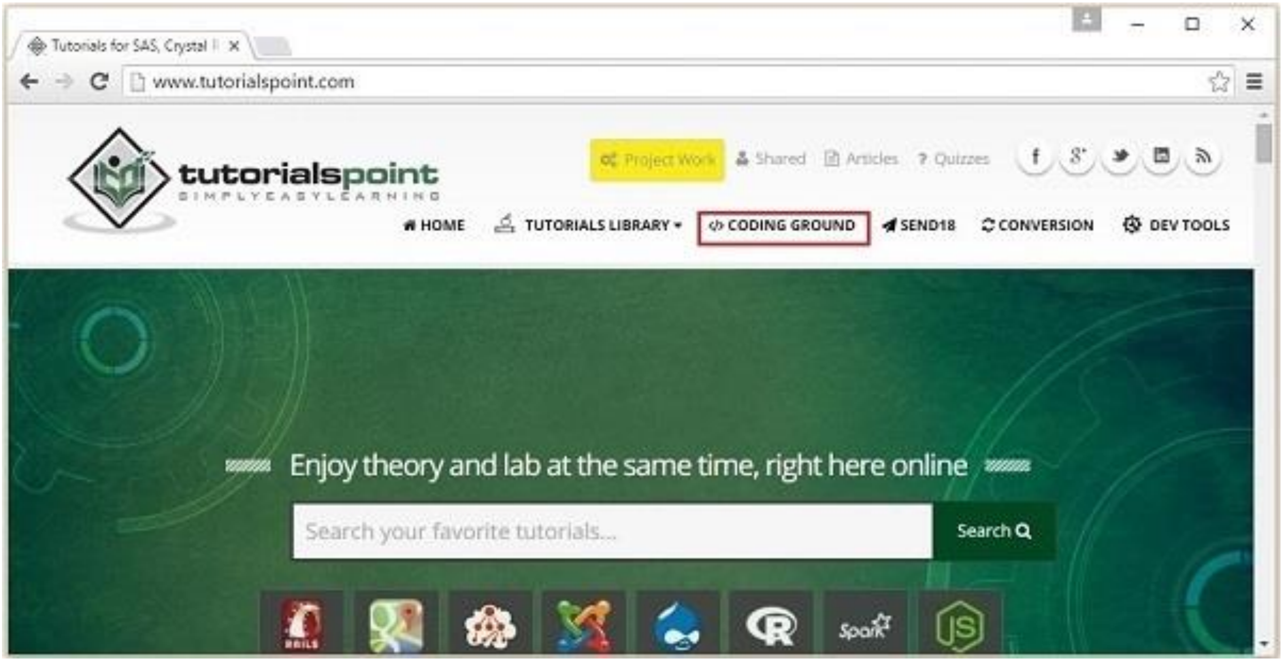
You can compile CoffeeScript files using Tutorials Point's CoffeeScript compiler provided in our Coding Ground section (http://www.tutorialspoint.com/codingground.htm). Follow the steps given below to use our CoffeeScript compiler.

Step 1

Click on the following link www.tutorialspoint.com. You will be directed to the homepage of our website.

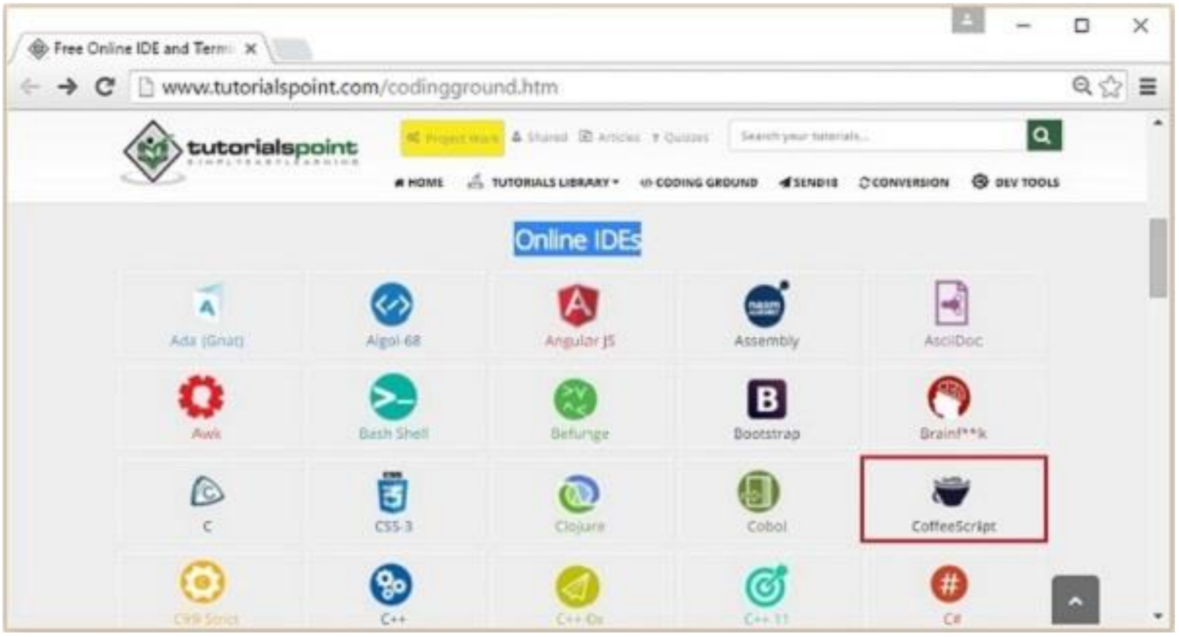
Step 2

Click on the button named **CODING GROUND** that is located at the top right corner of the homepage as highlighted in the snapshot given below.


www.tutorialspoint.com
HOME
TUTORIALS LIBRARY
CODING GROUND
SEND18
CONVERSION
DEV TOOLS
Enjoy theory and lab at the same time, right here online
Search your favorite tutorials...
Search

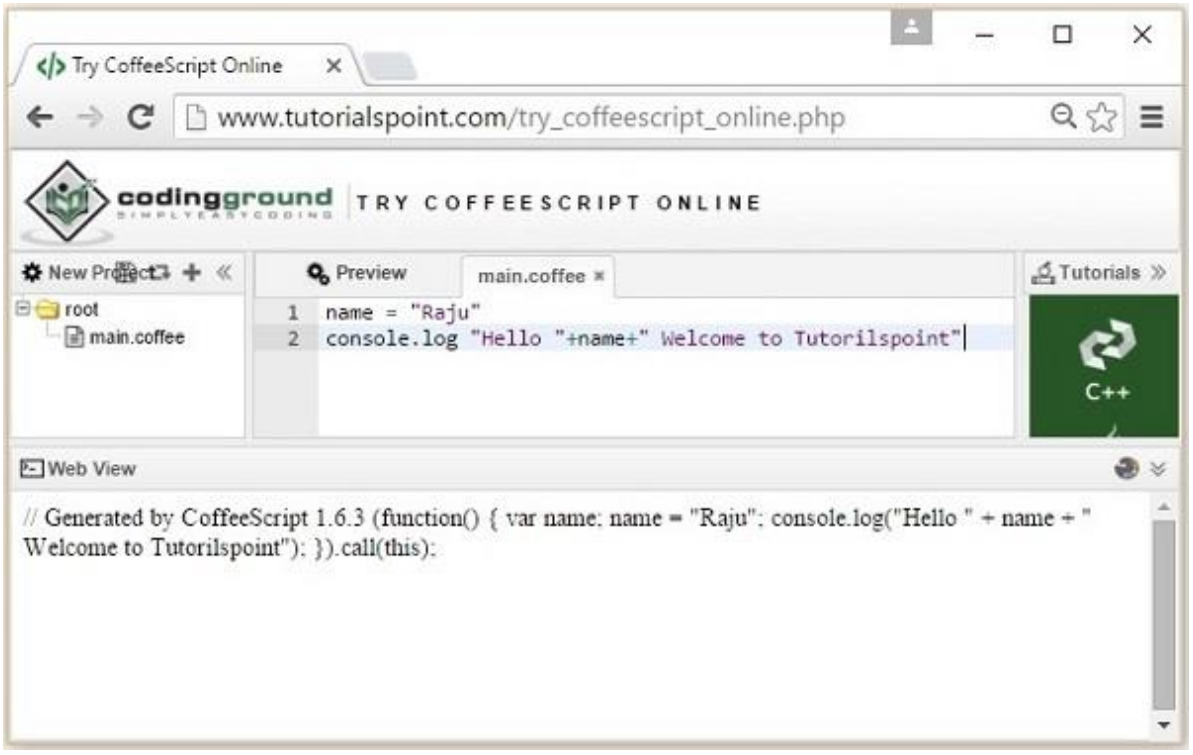
Step 3

This will lead to our **CODING GROUND** section which provides online terminals and IDEs for about 135 programming languages. Open CoffeeScript IDE in the Online IDEs section which is shown in the following snapshot.



Step 4

If you paste your CoffeeScript code in **main.coffee** (You can change the file name) and click the **Preview** button, then you can see the compiled JavaScript in the console as shown in the following snapshot.



```
name = "Raju"
console.log "Hello "+name+" Welcome to Tutorilspoint"
```

```
// Generated by CoffeeScript 1.6.3 (function() { var name; name = "Raju"; console.log("Hello " + name + " Welcome to Tutorilspoint"); }).call(this);
```

2. COFFEESCRIPT – ENVIRONMENT

The Compiler of the latest versions of CoffeeScript is written in CoffeeScript itself. To run CoffeeScript files in your system without a browser, you need a JavaScript runtime.

Node.js

Node.js is a JavaScript framework which is used to develop network server applications. It also acts as a bridge between JavaScript and the Operating System.

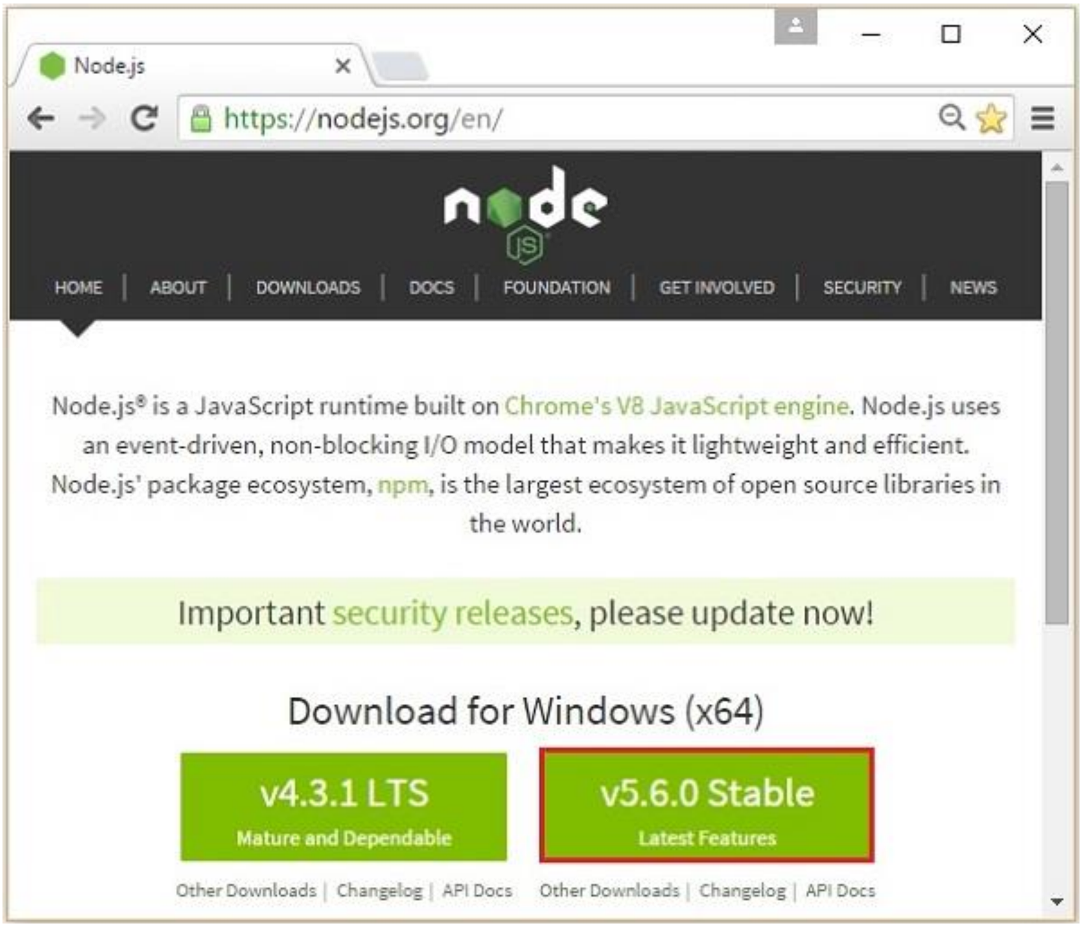
The command-line version of CoffeeScript is distributed as a Node.js package. Therefore, to install CoffeeScript (command-line) in your system, you first need install node.js.

Installing Node.js

Here are the steps to download and install Node.js in your system.

Step 1

Visit the nodejs homepage and download its stable version for windows by clicking on the button hilighted in the snapshot given below.


Node.js
https://nodejs.org/en/
HOME | ABOUT | DOWNLOADS | DOCS | FOUNDATION | GET INVOLVED | SECURITY | NEWS

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

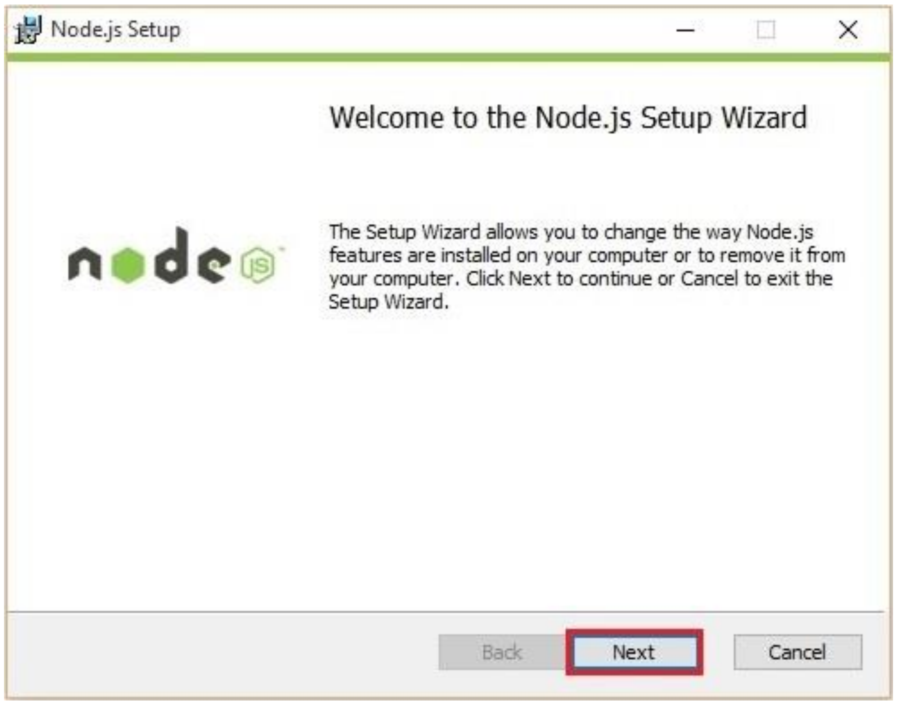
Important security releases, please update now!

Download for Windows (x64)

v4.3.1 LTS
Mature and Dependable
v5.6.0 Stable
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs

Step 2

On clicking, a *.msc* file named **node-v5.50-x64** will be downloaded into your system, run the downloaded file to start the Node.js set-up. Here is the snapshot of the Welcome page of No.js set-up wizard.

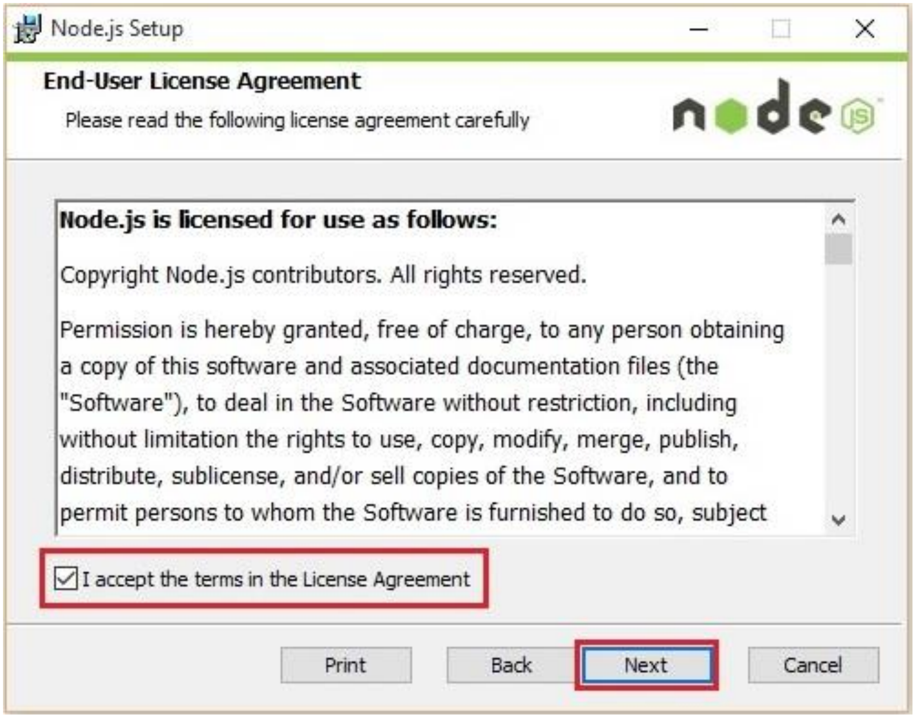


Welcome to the Node.js Setup Wizard

The Setup Wizard allows you to change the way Node.js features are installed on your computer or to remove it from your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step 3

Click on the Next button in the Welcome page of the Node.js set-up wizard which will lead you to the End-user License Agreement page. Accept the license agreement and click on the Next button as shown below.



End-User License Agreement

Please read the following license agreement carefully

Node.js is licensed for use as follows:

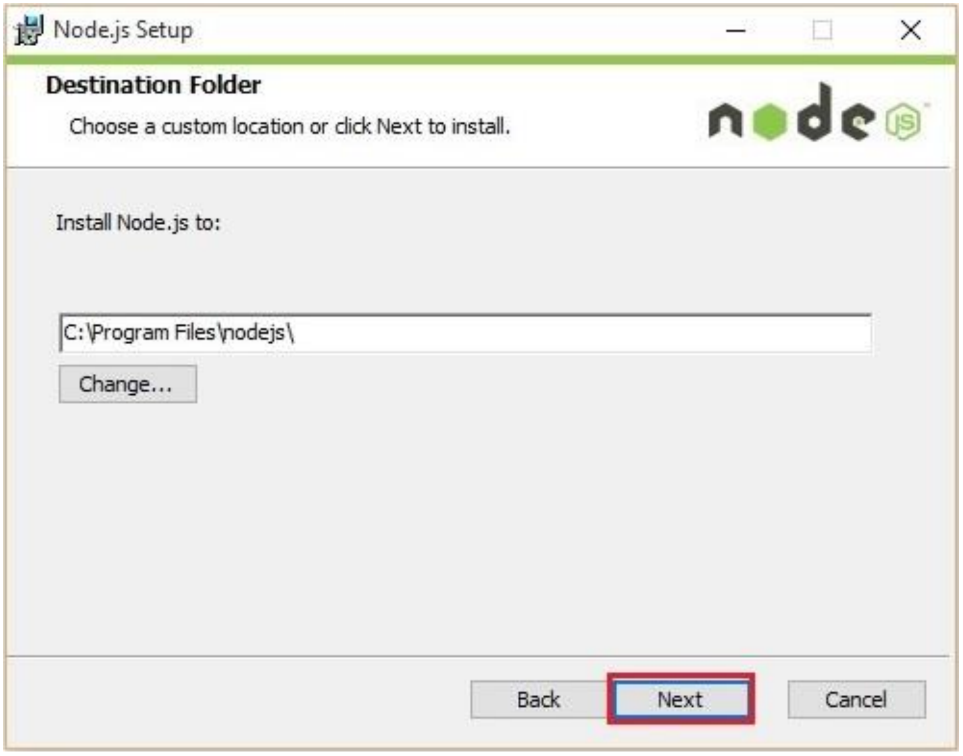
Copyright Node.js contributors. All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject

[✓] I accept the terms in the License Agreement

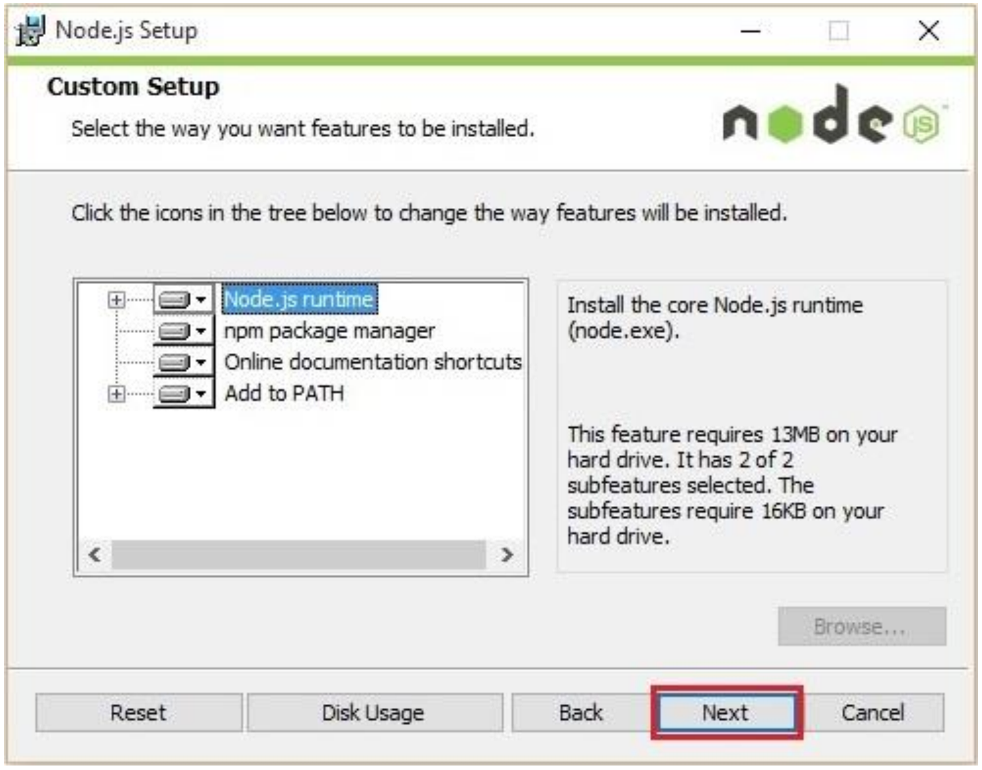
Step 4

On the next page, you need to set the destination folder to the path where you want to install Node.js. Change the path to the required folder and click on the Next button.



Step 5

In the **Custom setup** page, select the Node.js runtime to install node.exe file and click Next.



Custom Setup

Select the way you want features to be installed.

Click the icons in the tree below to change the way features will be installed.

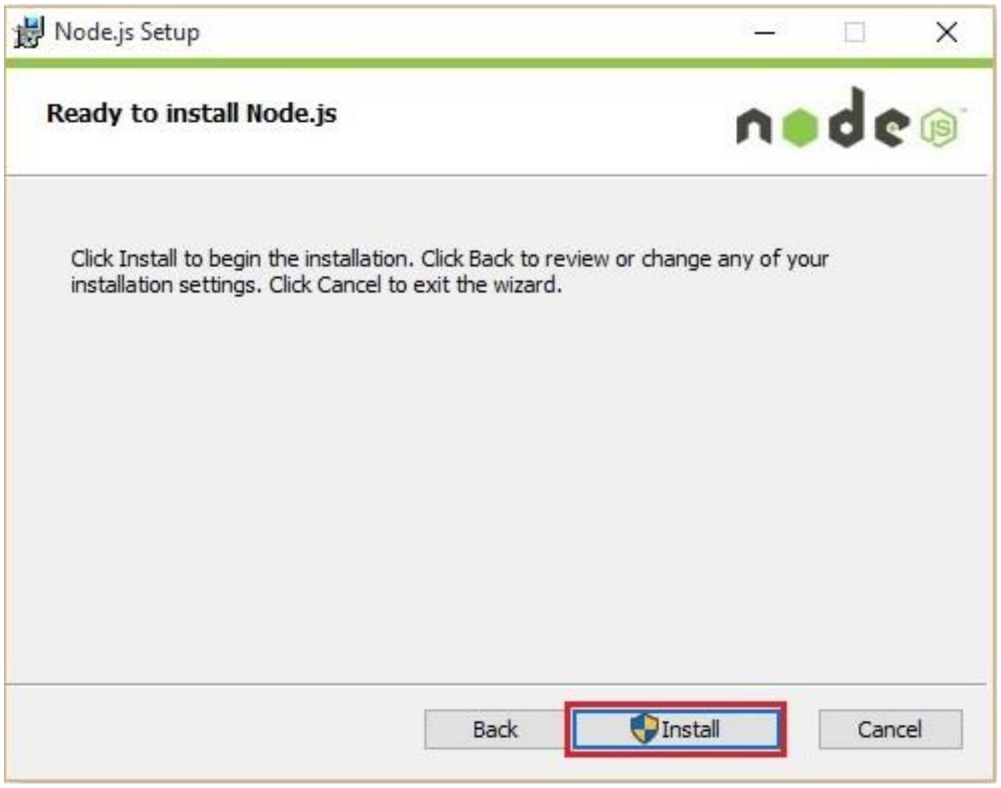
- Node.js runtime
- npm package manager
- Online documentation shortcuts
- Add to PATH

Install the core Node.js runtime (node.exe).

This feature requires 13MB on your hard drive. It has 2 of 2 subfeatures selected. The subfeatures require 16KB on your hard drive.

Step 6

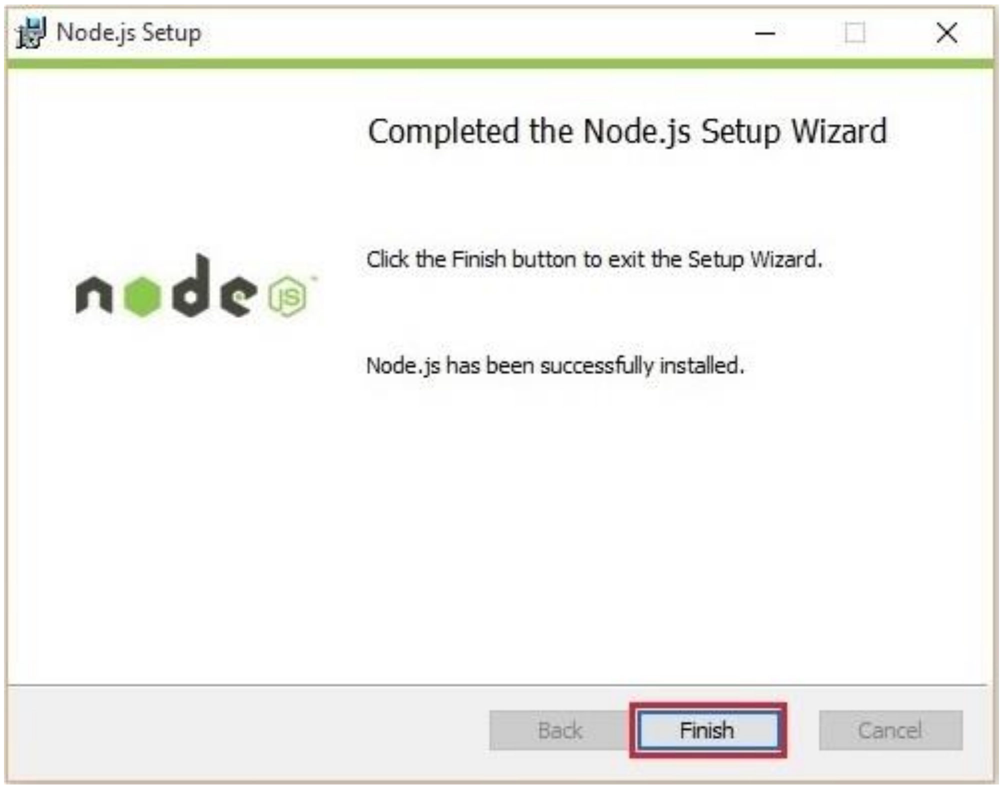
Finally, click on the Install button which will start the Node.js installation.


Node.js Setup

Ready to install Node.js

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Click on the Finish button of the Node.js set-up wizard as shown below to complete the Node.js installation.



Completed the Node.js Setup Wizard

Click the Finish button to exit the Setup Wizard.

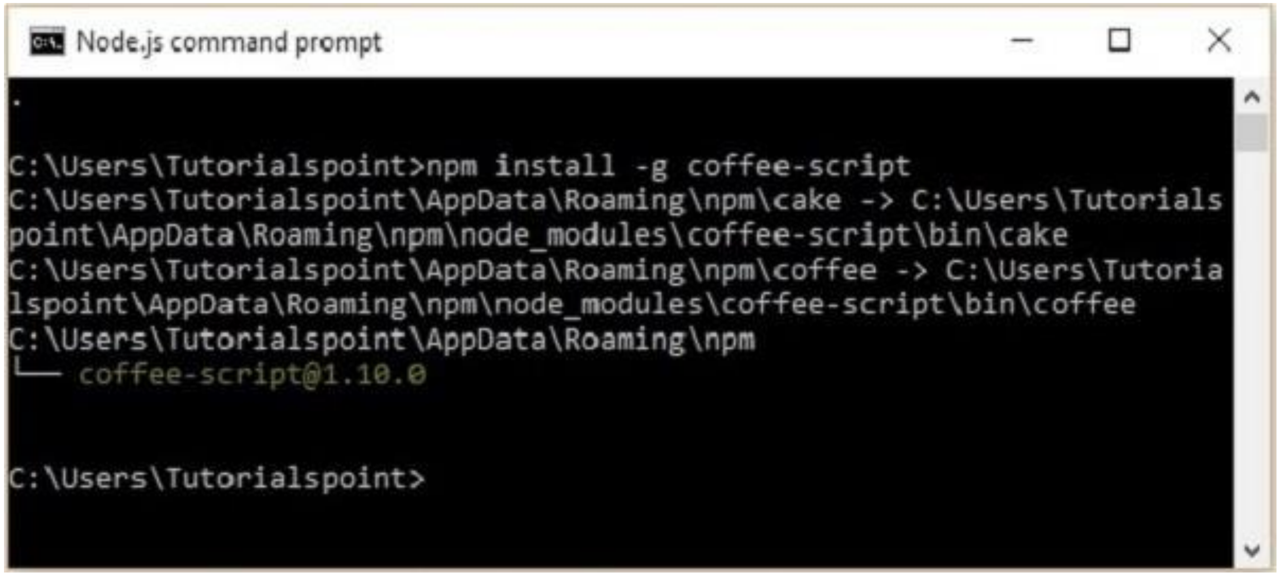
Node.js has been successfully installed.

Installing CoffeeScript

Node.js provides you a command prompt (**Node.js command prompt**). You can install CoffeeScript globally by entering the following command in it.

```
c:\> npm install -g coffeescript
```

On executing the the above command, CoffeeScript will be installed in your system by producing the following output



Node.js command prompt

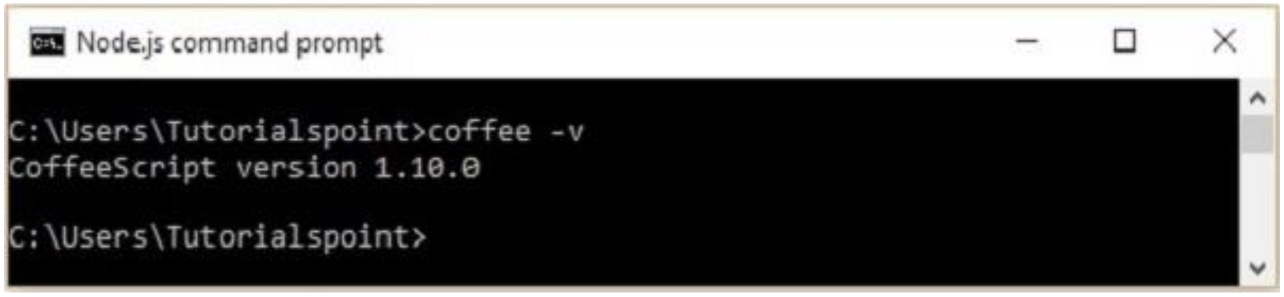
```
.

C:\Users\Tutorialspoint>npm install -g coffee-script
C:\Users\Tutorialspoint\AppData\Roaming\npm\cake -> C:\Users\Tutorials
point\AppData\Roaming\npm\node_modules\coffee-script\bin\cake
C:\Users\Tutorialspoint\AppData\Roaming\npm\coffee -> C:\Users\Tutoria
lspoint\AppData\Roaming\npm\node_modules\coffee-script\bin\coffee
C:\Users\Tutorialspoint\AppData\Roaming\npm
└── coffee-script@1.10.0


C:\Users\Tutorialspoint>
```

Verification

You can verify the installation of the CoffeeScript by typing the following command.

```
c:\> coffee -v
```

On successful installation, this command gives you the version of CoffeeScript as shown below.



Node.js command prompt

```
C:\Users\Tutorialspoint>coffee -v
CoffeeScript version 1.10.0

C:\Users\Tutorialspoint>
```

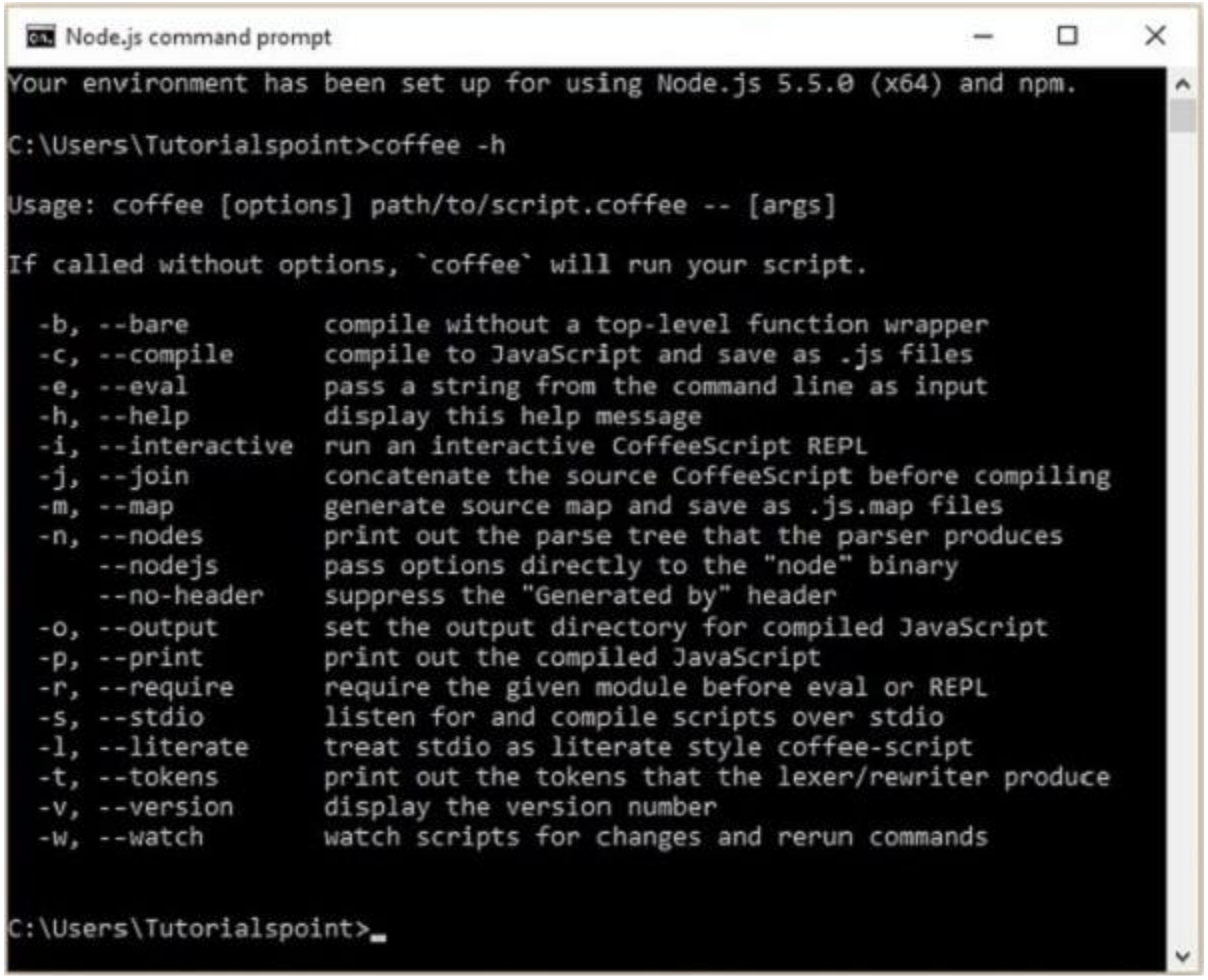
3. COFFEESCRIPT – COMMAND-LINE UTILITY

On installing CoffeeScript on Node.js, we can access the **coffee-command line utility**. In here, the **coffee** command is the key command. Using various options of this command, we can compile and execute the CoffeeScript files.

You can see the list of options of the **coffee** command using its **-h** or **--help** option. Open the **Node.js command prompt** and execute the following command in it.

```
c:\>coffee -help
```

This command gives you the list of various options of the **coffee**, along with the description of the operation performed by each of them as shown below.



```
Your environment has been set up for using Node.js 5.5.0 (x64) and npm.

C:\Users\Tutorialspoint>coffee -h

Usage: coffee [options] path/to/script.coffee -- [args]

If called without options, `coffee` will run your script.

  -b, --bare          compile without a top-level function wrapper
  -c, --compile       compile to JavaScript and save as .js files
  -e, --eval          pass a string from the command line as input
  -h, --help          display this help message
  -i, --interactive   run an interactive CoffeeScript REPL
  -j, --join          concatenate the source CoffeeScript before compiling
  -m, --map           generate source map and save as .js.map files
  -n, --nodes         print out the parse tree that the parser produces
      --nodejs        pass options directly to the "node" binary
      --no-header     suppress the "Generated by" header
  -o, --output        set the output directory for compiled JavaScript
  -p, --print         print out the compiled JavaScript
  -r, --require       require the given module before eval or REPL
  -s, --stdio         listen for and compile scripts over stdio
  -l, --literate      treat stdio as literate style coffee-script
  -t, --tokens        print out the tokens that the lexer/rewriter produce
  -v, --version       display the version number
  -w, --watch         watch scripts for changes and rerun commands


C:\Users\Tutorialspoint>_
```

Compiling the CoffeeScript Code

The CoffeeScript files are saved with the extension **.coffee**. You can compile these files using the **-c or --compile** option of the coffee command as shown below.

```
c:\>coffee -c filename.coffee
```

Example

Suppose there is a file in your system with the following CoffeeScript code which prints a message on the console.

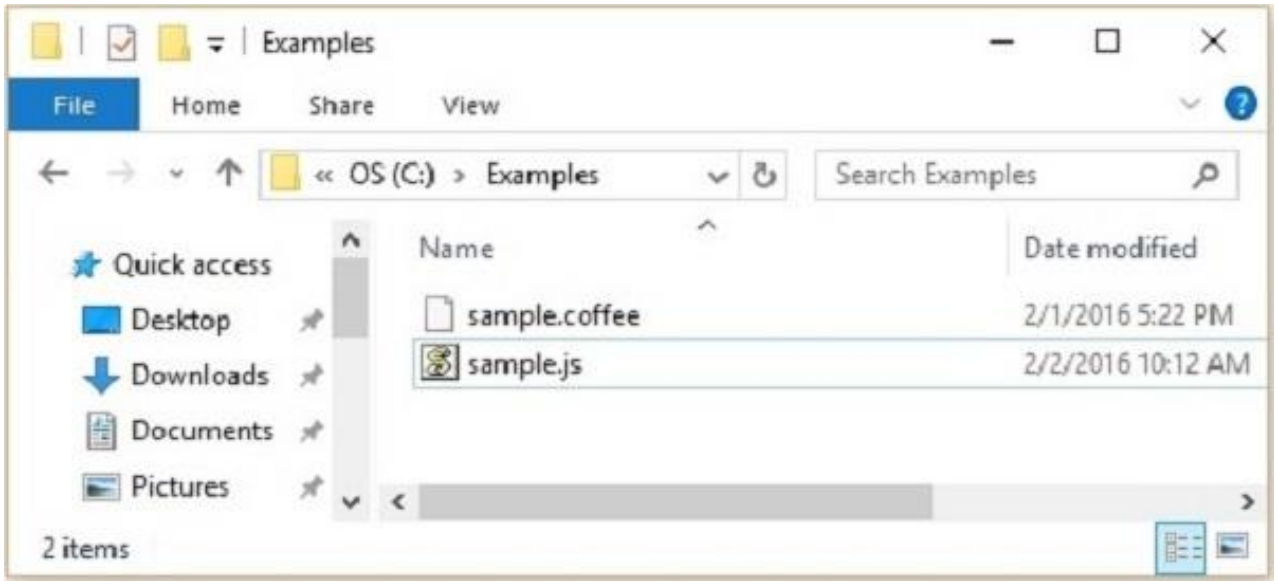
```
name = "Raju"

console.log "Hello"+name+" Welcome to Tutorilspoint"
```

Note – The **console.log()** function prints the given string on the consloe.

To compile the above code, save it in a file with the name **sample.coffee**. Open the Node.js command prompt. Browse through the path where you have saved the file and compile it using the **-c** option of the coffee command of the **coffee command-line utility** as shown below.

```
c:\> coffee -c sample.coffee
```

On executing the above command, the CoffeeScript compiler compiles the given file (sample.coffee) and saves it in the current location with a name sample.js as shown below.



```
Name                Date modified
sample.coffee       2/1/2016 5:22 PM
sample.js           2/2/2016 10:12 AM
```

If you open the sample.js file, you can observe the generated JavaScript as shown below.

```
// Generated by CoffeeScript 1.10.0
```

```
(function() {
  var name;
  name = "Raju";
  console.log("Hello " + name + " Welcome to Tutorilspoint");

}).call(this);
```

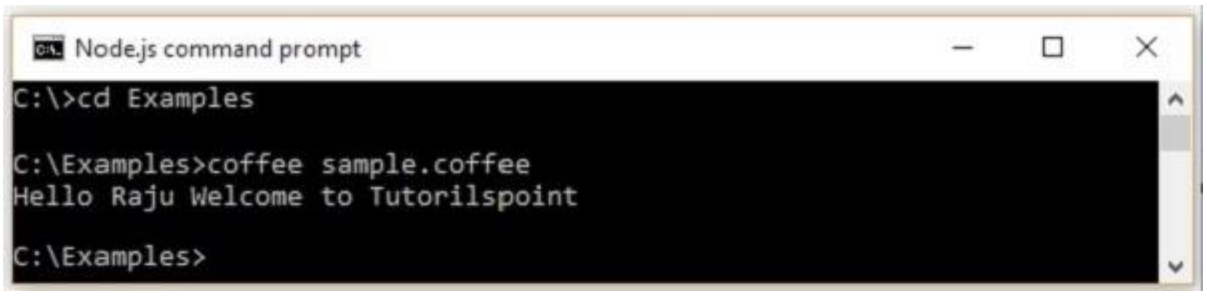
Executing the CoffeeScript code

You can execute a CoffeeScript file by simply passing the file name to the coffee command in the Node.js command prompt as follows.

```
c:\> coffee sample.coffee
```

Example

For example, let us execute the sample.coffee file. For this, open the Node.js command prompt. Browse through the path where you have saved the file and execute the file by directly passing its name to the coffee command as shown below.



```
C:\>cd Examples

C:\Examples>coffee sample.coffee
Hello Raju Welcome to Tutorilspoint

C:\Examples>
```

Watch and Compile

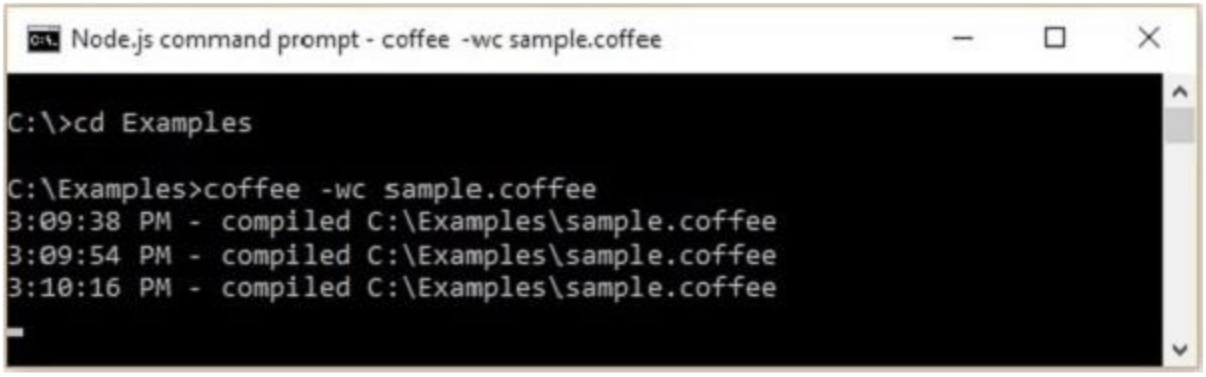
In some scenarios, there is a chance that we do a lot of changes to our scripts. Using the ***–w*** option of the coffee command, you watch your scripts for changes.

You can watch and compile a file simultaneously using the ***-wc*** option as shown below. When we use this option, the file will be recompiled each time you make changes in your script.

```
c:\>coffee -wc file_name
```

Example

Suppose we have compiled a file named **sample.coffee** using the **-wc** option and we modified the script thrice. Each time we change the script, the **.coffee** file is recompiled leaving the Node.js command prompt as shown below.



```
C:\>cd Examples

C:\Examples>coffee -wc sample.coffee
3:09:38 PM - compiled C:\Examples\sample.coffee
3:09:54 PM - compiled C:\Examples\sample.coffee
3:10:16 PM - compiled C:\Examples\sample.coffee
```

Setting the Output Directory

Using the ***-o*** option, we can set the output directory to place the compiled JavaScript files as shown below.

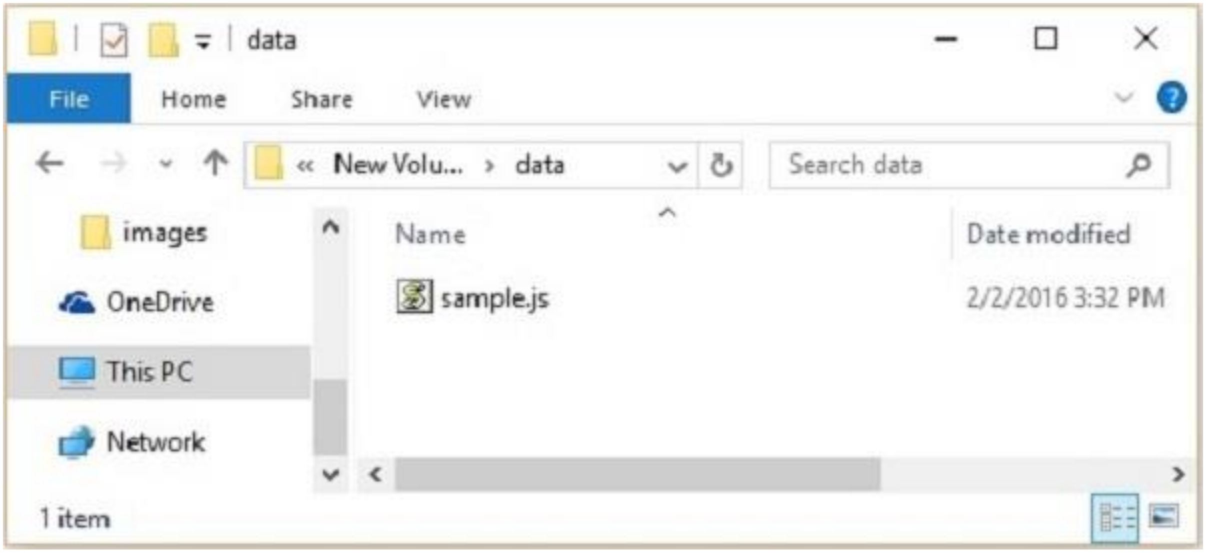
```
c:\>coffee -o "Required path where we want our .js files" file_name
```

Example

Let us save the JavaScript code of the sample.coffee file in a folder named **data** in the E drive using the ***-o*** option by executing the following command in the command prompt.

```
c:\>coffee -o E://data sample.coffee
```

Following is the snapshot of the given folder after executing the above command. Here you can observe the JavaScript file of the sample.coffee



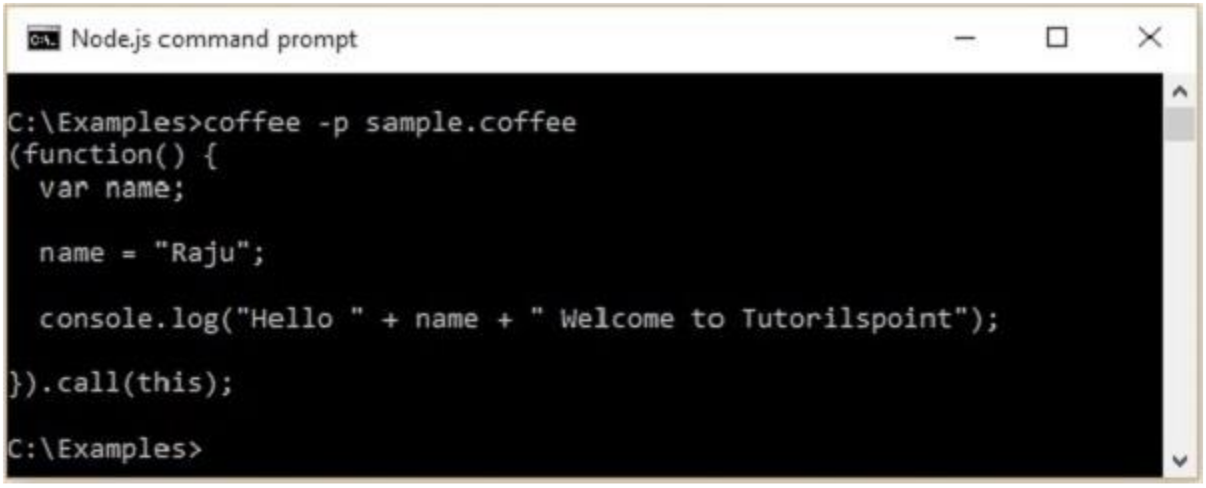
Print the Compiled JavaScript

If we want to print the compiled javascript on the console itself, we have to use the ***-p*** option of the coffee command as shown below.

```
c:\>coffee -p file_name
```

Example

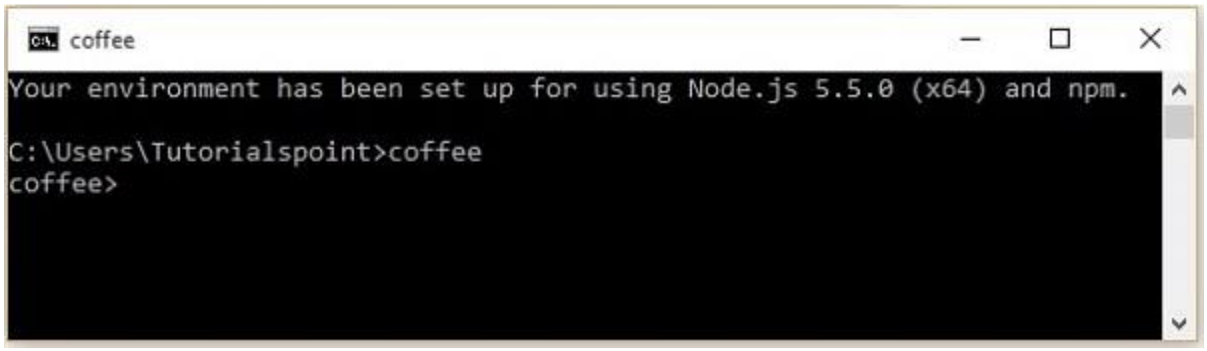
For example, you can print the compiled JavaScript code of the *sample.coffee* file on the console using the *-p* option as shown below.



```
C:\Examples>coffee -p sample.coffee
(function() {
  var name;

  name = "Raju";

  console.log("Hello " + name + " Welcome to Tutorilspoint");

}).call(this);

C:\Examples>
```

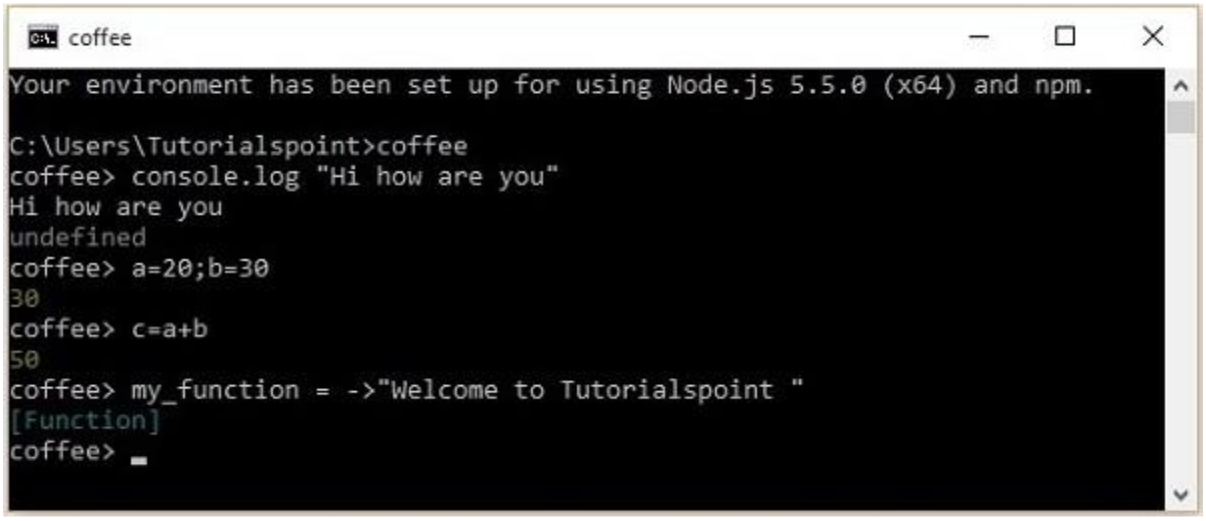
The REPL (Read Evaluate Print Loop)

CoffeeScript provides you an REPL-interactive shell. This shell is used to evaluate the CoffeeScript expressions. You can type any CoffeeScript code in this shell and get the result immediately. You can open REPL by executing the **coffee** command without any options as shown below.



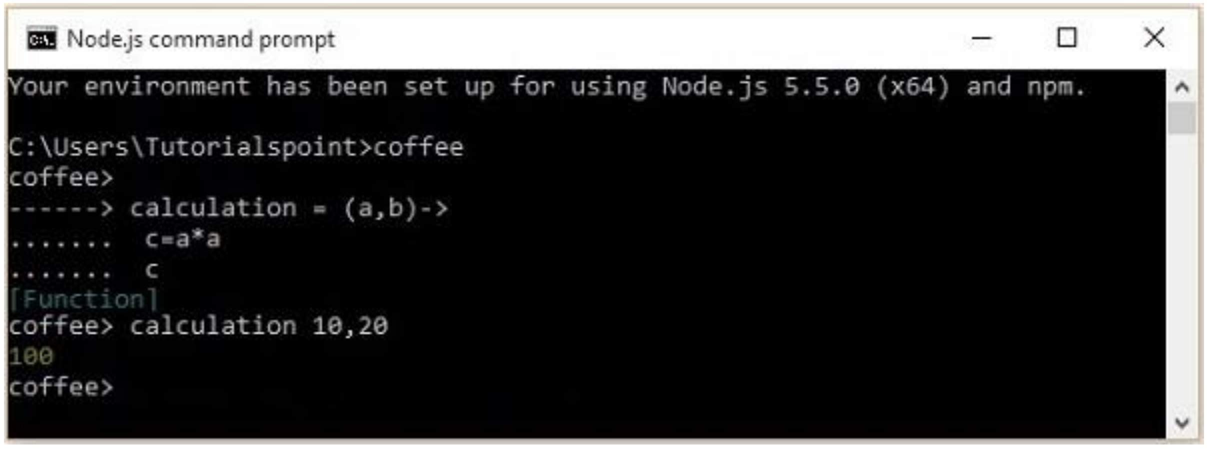
```
Your environment has been set up for using Node.js 5.5.0 (x64) and npm.

C:\Users\Tutorialspoint>coffee
coffee>
```

Using this shell, we can assign values to variables, create functions, and evaluate results. As shown in the following screenshot, if we call functions in REPL, it prints the value of the function. If we give an expression to it, it evaluates and prints the result of the expression. And if we simply type the statements in it, it prints the value of the last statement.



```
Your environment has been set up for using Node.js 5.5.0 (x64) and npm.

C:\Users\Tutorialspoint>coffee
coffee> console.log "Hi how are you"
Hi how are you
undefined
coffee> a=20;b=30
30
coffee> c=a+b
50
coffee> my_function = ->"Welcome to Tutorialspoint "
[Function]
coffee> _
```

In RPEL, you can access multiple line mode by pressing *ctrl+v* where you can evaluate the code with multiple lines (like functions) and you can get back to REPL mode from it by pressing *ctrl+v* again. Here is an example usage of the multi line mode.



```
Your environment has been set up for using Node.js 5.5.0 (x64) and npm.

C:\Users\Tutorialspoint>coffee
coffee>
------> calculation = (a,b)->
.......  c=a*a
.......  c
[Function]
coffee> calculation 10,20
100
coffee>
```

Running CoffeeScript through Browser

We can run CoffeeScript using the <script> tag of the HTML just like JavaScript as shown below.

```
<script src="http://jashkenas.github.com/coffee-script/extras/coffee-script.js"
type="text/javascript" charset="utf-8"></script>

<script type="text/coffeescript">

  # Some CoffeeScript

</script>
```

But for this, we have to import the library in each application and the CoffeeScript code will be interpreted line by line before the output is shown. This will slow down your applications, therefore this approach is not recommended.

Therefore, to use CoffeeScript in your applications, you need to pre-compile them using the Coffee command-line utility and then you can use the generated JavaScript in your applications.