



# ES6

**tutorialspoint**

SIMPLY EASY LEARNING

[www.tutorialspoint.com](http://www.tutorialspoint.com)



<https://www.facebook.com/tutorialspointindia>



<https://twitter.com/tutorialspoint>

## About the Tutorial

---

ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. Languages like JavaScript, Jscript and ActionScript are governed by this specification.

This tutorial introduces you to ES6 implementation in JavaScript.

## Audience

---

This tutorial has been prepared for JavaScript developers who are keen on knowing the difference between ECMAScript 5 and ECMAScript 6. It is useful for those who want to learn the latest developments in the language and implement the same in JavaScript.

## Prerequisites

---

You need to have a basic understanding of JavaScript to make the most of this tutorial.

## Disclaimer & Copyright

---

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at [contact@tutorialspoint.com](mailto:contact@tutorialspoint.com).

# Table of Contents

---

About the Tutorial.....	i
Audience .....	i
Prerequisites .....	i
Disclaimer & Copyright.....	i
Table of Contents .....	ii
 1. ES6 – OVERVIEW .....	 1
JavaScript .....	1
 2. ES6 – ENVIRONMENT .....	 2
Local Environment Setup.....	2
Installation on Windows .....	2
Installation on Mac OS X .....	3
Installation on Linux.....	4
Integrated Development Environment (IDE) Support .....	4
Visual Studio Code .....	4
Brackets .....	7
 3. ES6 – SYNTAX .....	 10
Whitespace and Line Breaks.....	11
Comments in JavaScript .....	11
Your First JavaScript Code .....	12
Executing the Code.....	12
Node.js and JS/ES6 .....	13
The Strict Mode.....	13
ES6 and Hoisting.....	14

4.	<a href="#">ES6 – VARIABLES</a>	15
	Type Syntax	15
	JavaScript and Dynamic Typing	16
	JavaScript Variable Scope	16
	The Let and Block Scope	17
	The const	18
	ES6 and Variable Hoisting	18
5.	ES6 – OPERATORS	20
	Arithmetic Operators	20
	Relational Operators	22
	Logical Operators	23
	Bitwise Operators	24
	Assignment Operators	26
	Miscellaneous Operators	27
	Type Operators	29
6.	ES6 – DECISION MAKING	30
	The if Statement	30
	The if...else Statement	32
	The else...if Ladder	33
	The switch...case Statement	34
7.	ES6 – LOOPS	38
	Definite Loop	38
	Indefinite Loop	41
	The Loop Control Statements	44
	Using Labels to Control the Flow	45

8.	ES6 – FUNCTIONS .....	49
	Classification of Functions .....	49
	Rest Parameters .....	52
	Anonymous Function .....	52
	The Function Constructor .....	53
	Recursion and JavaScript Functions .....	54
	Lambda Functions .....	55
	Function Expression and Function Declaration .....	56
	Function Hoisting .....	57
	Immediately Invoked Function Expression .....	57
	Generator Functions .....	58
9.	ES6 – EVENTS .....	61
	Event Handlers .....	61
	onclick Event Type .....	61
	onsubmitEvent Type .....	62
	onmouseover and onmouseout .....	63
	HTML 5 Standard Events .....	63
10.	ES6 – COOKIES .....	70
	How It Works? .....	70
	Storing Cookies .....	70
	Reading Cookies .....	72
	Setting Cookies Expiry Date .....	73
	Deleting a Cookie .....	74
11.	ES6 – PAGE REDIRECT .....	76
	JavaScript Page Redirection .....	76
	Redirection and Search Engine Optimization .....	77

12. ES6 – DIALOG BOXES .....	79
Alert Dialog Box .....	79
Confirmation Dialog Box .....	80
Prompt Dialog Box .....	81
13. ES6 – VOID KEYWORD .....	83
Void and Immediately Invoked Function Expressions .....	83
Void and JavaScript URIs .....	83
14. ES6 – PAGE PRINTING.....	85
15. ES6 – OBJECTS .....	86
Object Initializers .....	86
The Object() Constructor .....	87
Constructor Function.....	89
The Object.create Method .....	91
The Object.assign() Function .....	91
Deleting Properties .....	93
Comparing Objects.....	93
Object De-structuring.....	94
16. ES6 – NUMBER .....	95
Number Properties.....	95
EPSILON .....	96
MAX_SAFE_INTEGER .....	96
MAX_VALUE.....	96
MIN_SAFE_INTEGER.....	97
MIN_VALUE.....	97
Nan .....	98

NEGATIVE_INFINITY .....	98
POSITIVE_INFINITY .....	99
Number Methods .....	99
Number.isNaN() .....	100
Number.isFinite.....	100
Number.isInteger().....	101
Number.isSafeInteger() .....	101
Number.parseInt() .....	102
Number.parseFloat().....	102
Number Instances Methods .....	103
toExponential() .....	103
toFixed().....	104
toLocaleString() .....	105
toPrecision().....	105
toString().....	106
valueOf() .....	107
Binary and Octal Literals .....	107
17. ES6 – BOOLEAN .....	109
Boolean Properties.....	109
Boolean Methods.....	111
toSource () .....	111
toString () .....	112
valueOf () .....	113
18. ES6 – STRINGS .....	114
String Properties .....	114
Constructor .....	114

Length .....	115
Prototype .....	115
String Methods .....	116
charAt .....	117
charCodeAt() .....	118
concat() .....	119
indexOf() .....	119
lastIndexOf() .....	120
localeCompare() .....	121
replace() .....	122
search() .....	123
slice() .....	124
split() .....	125
substr() .....	125
substring() .....	126
toLocaleLowerCase() .....	127
toLowerCase() .....	127
toString() .....	128
toUpperCase() .....	128
valueOf() .....	129
19. ES6 – NEW STRING METHODS .....	130
startsWith .....	130
endsWith.....	131
includes() .....	131
repeat() .....	132
Template Literals.....	133



Multiline Strings and Template Literals .....	134
String.raw() .....	134
String.fromCodePoint() .....	135
20. ES6 – ARRAYS .....	136
Features of an Array.....	136
Declaring and Initializing Arrays .....	136
Accessing Array Elements.....	137
Array Object.....	138
Array Methods .....	139
concat().....	140
every().....	141
filter().....	141
forEach() .....	142
indexOf() .....	143
join() .....	144
lastIndexOf() .....	145
map() .....	146
pop() .....	146
push().....	147
reduce().....	148
reduceRight() .....	148
reverse().....	149
shift() .....	150
slice() .....	150
some().....	151
sort() .....	152

splice()	152
toString()	153
unshift()	154
ES6 – Array Methods	154
Array Traversal using for...in loop	157
Arrays in JavaScript	157
Array De-structuring	160
<b>21. ES6 – DATE</b>	<b>161</b>
Date Properties	161
Constructor	161
prototype	162
Date Methods	163
Date()	165
getDate()	165
getDay()	166
getFullYear()	166
getHours()	167
getMilliseconds()	167
getMinutes()	168
getMonth()	168
getSeconds()	169
getTime()	169
getTimezoneOffset()	170
getUTCDate()	170
getUTCDay()	171
getUTCFullYear()	171

<b>getUTCHours()</b> .....	<b>172</b>
<b>getUTCMilliseconds()</b> .....	<b>172</b>
<b>getUTCMinutes()</b> .....	<b>173</b>
<b>getUTCMonth()</b> .....	<b>173</b>
<b>getUTCSeconds()</b> .....	<b>174</b>
<b>setDate()</b> .....	<b>174</b>
<b>setFullYear()</b> .....	<b>175</b>
<b>setHours()</b> .....	<b>175</b>
<b>setMilliseconds()</b> .....	<b>176</b>
<b>setMinutes()</b> .....	<b>177</b>
<b>setMonth()</b> .....	<b>177</b>
<b>setSeconds()</b> .....	<b>178</b>
<b>setTime()</b> .....	<b>179</b>
<b>setUTCDate()</b> .....	<b>180</b>
<b>setUTCFullYear()</b> .....	<b>180</b>
<b>setUTCHours()</b> .....	<b>181</b>
<b>setUTCMilliseconds()</b> .....	<b>182</b>
<b>setUTCMinutes()</b> .....	<b>182</b>
<b>setUTCMonth()</b> .....	<b>183</b>
<b>setUTCSeconds()</b> .....	<b>184</b>
<b>toDatestring()</b> .....	<b>184</b>
<b>toLocaleDateString()</b> .....	<b>185</b>
<b>toLocaleString()</b> .....	<b>185</b>
<b>toLocaleTimeString()</b> .....	<b>186</b>
<b>toString()</b> .....	<b>187</b>
<b>toTimeString()</b> .....	<b>187</b>
<b>toUTCString()</b> .....	<b>188</b>

<b>valueOf()</b> .....	<b>188</b>
<b>22. ES6 – MATH</b> .....	<b>190</b>
<b>Math Properties</b> .....	<b>190</b>
<b>Math- E</b> .....	<b>190</b>
<b>Math- LN2</b> .....	<b>190</b>
<b>Math- LN10</b> .....	<b>191</b>
<b>Math- LOG2E</b> .....	<b>191</b>
<b>Math - LOG10E</b> .....	<b>192</b>
<b>Math- PI</b> .....	<b>192</b>
<b>Math- SQRT1_2</b> .....	<b>192</b>
<b>Math - SQRT2</b> .....	<b>193</b>
<b>Exponential Functions</b> .....	<b>193</b>
<b>Pow()</b> .....	<b>193</b>
<b>sqrt()</b> .....	<b>194</b>
<b>cbrt()</b> .....	<b>195</b>
<b>exp()</b> .....	<b>195</b>
<b>expm1(x)</b> .....	<b>196</b>
<b>Math.hypot(x1, x2,...)</b> .....	<b>197</b>
<b>Logarithmic Functions</b> .....	<b>197</b>
<b>Math.log(x)</b> .....	<b>198</b>
<b>Math.log10(x)</b> .....	<b>198</b>
<b>Math.log2(x)</b> .....	<b>199</b>
<b>Math.log1p(x)</b> .....	<b>199</b>
<b>Miscellaneous Algebraic Functions</b> .....	<b>200</b>
<b>Abs()</b> .....	<b>200</b>
<b>sign()</b> .....	<b>201</b>

<b>round()</b> .....	<b>202</b>
<b>trunc()</b> .....	<b>202</b>
<b>floor()</b> .....	<b>203</b>
<b>ceil()</b> .....	<b>203</b>
<b>min()</b> .....	<b>204</b>
<b>max()</b> .....	<b>205</b>
<b>Trigonometric Functions</b> .....	<b>205</b>
<b>Math.sin(x)</b> .....	<b>206</b>
<b>Math.cos(x)</b> .....	<b>206</b>
<b>Math.tan(x)</b> .....	<b>207</b>
<b>Math.asin(x)</b> .....	<b>207</b>
<b>Math.acos(x)</b> .....	<b>208</b>
<b>Math.atan(x)</b> .....	<b>209</b>
<b>Math.atan2()</b> .....	<b>209</b>
<b>Math.random()</b> .....	<b>210</b>
<b>23. ES6 – REGEXP</b> .....	<b>211</b>
<b>Constructing Regular Expressions</b> .....	<b>211</b>
<b>Meta-characters</b> .....	<b>214</b>
<b>RegExp Properties</b> .....	<b>215</b>
<b>RegExp Constructor</b> .....	<b>215</b>
<b>global</b> .....	<b>216</b>
<b>ignoreCase</b> .....	<b>217</b>
<b>lastIndex</b> .....	<b>218</b>
<b>multiline</b> .....	<b>218</b>
<b>source</b> .....	<b>219</b>
<b>RegExp Methods</b> .....	<b>220</b>

exec()	220
test()	221
match()	222
replace()	222
search()	223
split()	224
toString()	225
<b>24. ES6 – HTML DOM</b>	<b>226</b>
The Legacy DOM	227
Document Properties in Legacy DOM	227
Document Methods in Legacy DOM	229
<b>25. ES6 – COLLECTIONS</b>	<b>232</b>
Maps	232
Understanding basic Map operations	233
Map Methods	235
clear()	235
delete(key)	236
entries()	237
forEach	237
keys()	238
values()	239
The for...of Loop	239
Weak Maps	240
Sets	240
Set Properties	241
Set Methods	241

add()	242
clear()	243
delete()	243
entries()	244
forEach	245
has()	245
values() and keys()	246
Weak Set	248
Iterator	248
26. ES6 – CLASSES	251
Object-Oriented Programming Concepts	251
Creating Objects	253
Accessing Functions	253
The Static Keyword	254
The instanceof operator	255
Class Inheritance	255
Class Inheritance and Method Overriding	257
The Super Keyword	257
27. ES6 – PROMISES	259
Understanding Callback	259
Understanding AsyncCallback	260
28. ES6 – MODULES	266
Exporting a Module	266
Importing a Module	266

29. ES6 – ERROR HANDLING .....	269
Syntax Errors .....	269
Runtime Errors .....	269
Logical Errors.....	269
Throwing Exceptions .....	270
Exception Handling .....	270
The onerror( ) Method .....	272
Custom Errors .....	273
30. ES6 – VALIDATIONS .....	275
Basic Form Validation.....	277
Data Format Validation .....	278
31. ES6 – ANIMATION .....	279
Manual Animation .....	280
Automated Animation .....	281
Rollover with a Mouse Event.....	282
32. ES6 – MULTIMEDIA .....	284
Checking for Plugins .....	285
Controlling Multimedia .....	286
33. ES6 – DEBUGGING.....	288
Error Messages in IE .....	288
Error Messages in Firefox or Mozilla .....	289
Error Notifications.....	289
Debugging a Script .....	289
Useful Tips for Developers .....	290
Debugging with Node.js .....	291



Visual Studio Code and Debugging .....	292
34. ES6 – IMAGE MAP .....	293
35. ES6 – BROWSERS.....	295
Navigator Properties .....	295
Navigator Methods .....	296
Browser Detection .....	296

# 1. ES6 – Overview

ECMAScript (ES) is a scripting language specification standardized by ECMAScript International. It is used by applications to enable client-side scripting. The specification is influenced by programming languages like Self, Perl, Python, Java etc. Languages like JavaScript, Jscript and ActionScript are governed by this specification.

This tutorial introduces you to ES6 implementation in JavaScript.

## JavaScript

---

JavaScript was developed by Brendan Eich, a developer at Netscape Communications Corporation, in 1995. JavaScript started life with the name Mocha, and was briefly named LiveScript before being officially renamed to JavaScript. It is a scripting language that is executed by the browser, i.e. on the client's end. It is used in conjunction with HTML to develop responsive webpages.

ECMA Script6's implementation discussed here covers the following new features:

- Support for constants
- Block Scope
- Arrow Functions
- Extended Parameter Handling
- Template Literals
- Extended Literals
- Enhanced Object Properties
- De-structuring Assignment
- Modules
- Classes
- Iterators
- Generators
- Collections
- New built in methods for various classes
- Promises

## 2. ES6 – Environment

In this chapter, we will discuss the setting up of the environment for ES6.

### Local Environment Setup

---

JavaScript can run on any browser, any host, and any OS. You will need the following to write and test a JavaScript program standard:

#### Text Editor

The text editor helps you to write your source code. Examples of few editors include Windows Notepad, Notepad++, Emacs, vim or vi etc. Editors used may vary with the operating systems. The source files are typically named with the **extension .js**.

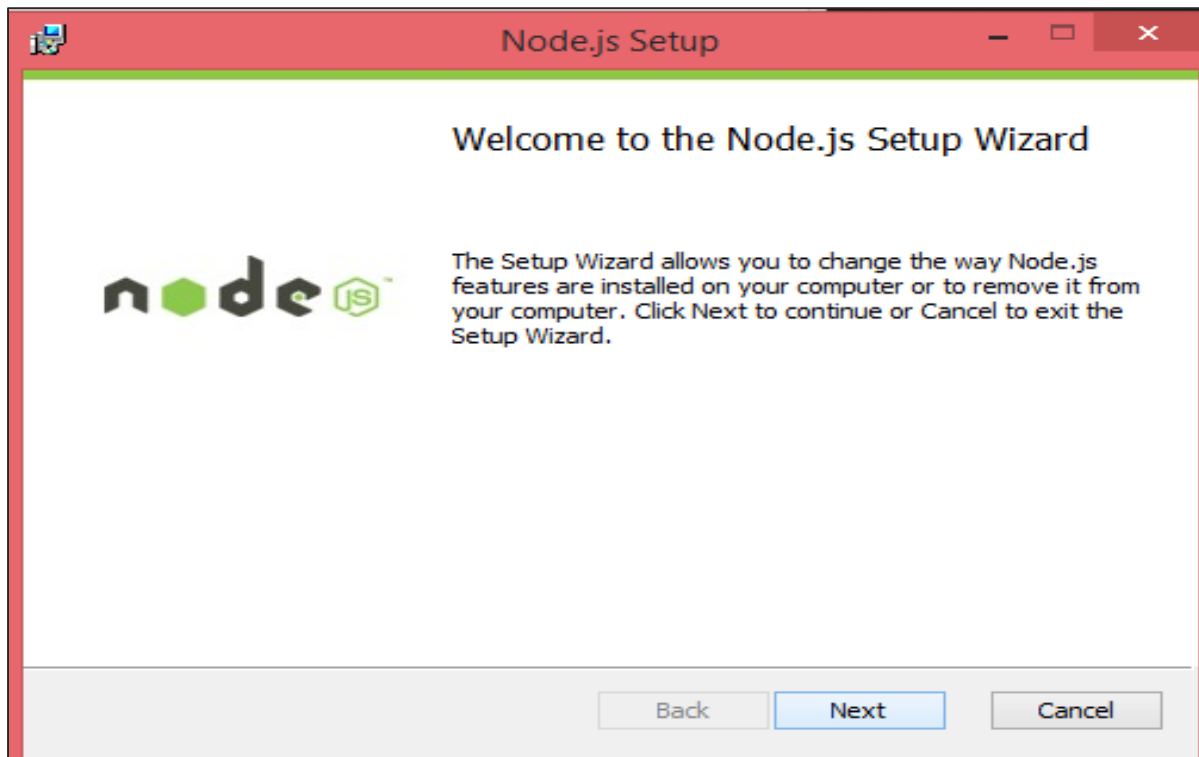
#### Installing Node.js

**Node.js** is an open source, cross-platform runtime environment for server-side JavaScript. Node.js is required to run JavaScript without a browser support. It uses Google V8 JavaScript engine to execute the code. You may download Node.js source code or a pre-built installer for your platform. Node is available at <https://nodejs.org/en/download>

#### Installation on Windows

---

Download and run the **.msi installer** for Node.



To verify if the installation was successful, enter the command **node -v** in the terminal window.

```
C:\Users>node -v
v4.2.3
C:\Users>_
```

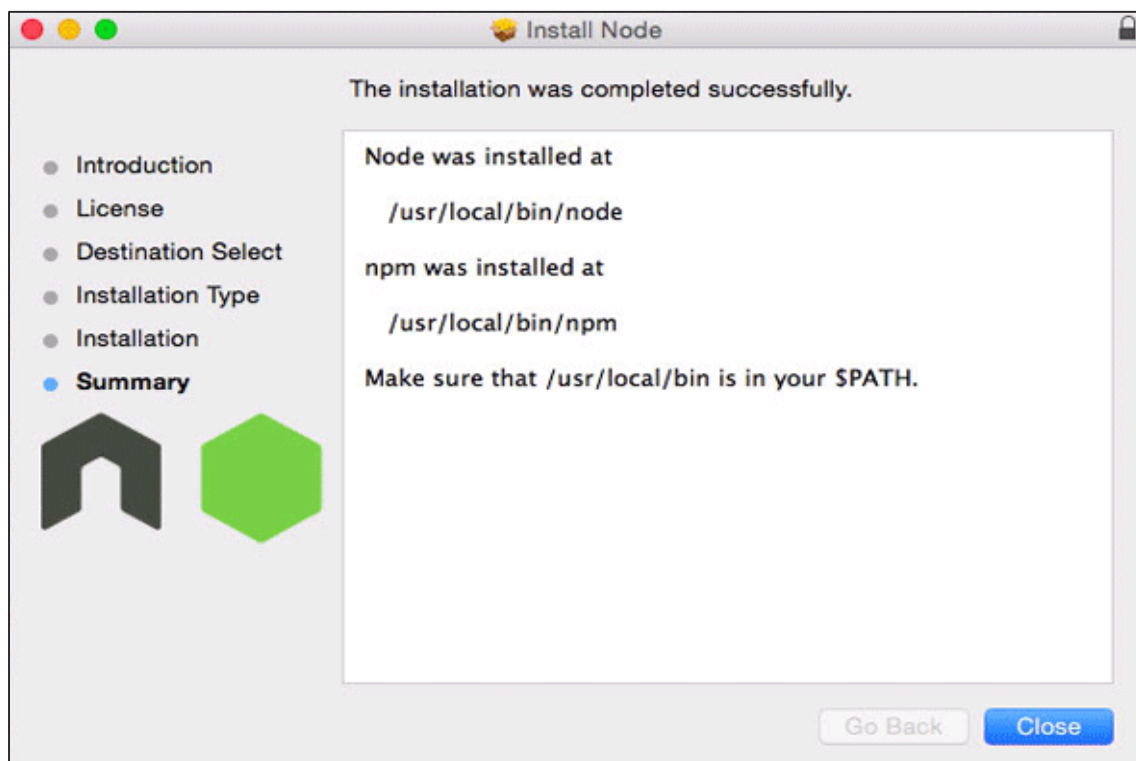
## Installation on Mac OS X

---

To install node.js on OS X you can download a pre-compiled binary package which makes a nice and easy installation. Head over to <http://nodejs.org/> and click the install button to download the latest package.



Install the package from the **.dmg** by following along the install wizard which will install both **node** and **npm**. npm is the Node Package Manager which facilitates installs of additional packages for Node.js.



## Installation on Linux

---

You need to install a number of **dependencies** before you can install Node.js and npm.

- **Ruby** and **GCC**. You'll need Ruby 1.8.6 or newer and GCC 4.2 or newer.
- **Homebrew**. Homebrew is a package manager originally for the Mac, but it's been ported to Linux as Linuxbrew. You can learn more about Homebrew at the <http://brew.sh> and Linuxbrew at the <http://brew.sh/linuxbrew>.

## Integrated Development Environment (IDE) Support

---

JavaScript can be built on a plethora of development environments like Visual Studio, Sublime Text 2, WebStorm/PHPStorm, Eclipse, Brackets, etc. The Visual Studio Code and Brackets IDE is discussed in this section. The development environment used here is Visual Studio Code (Windows platform).

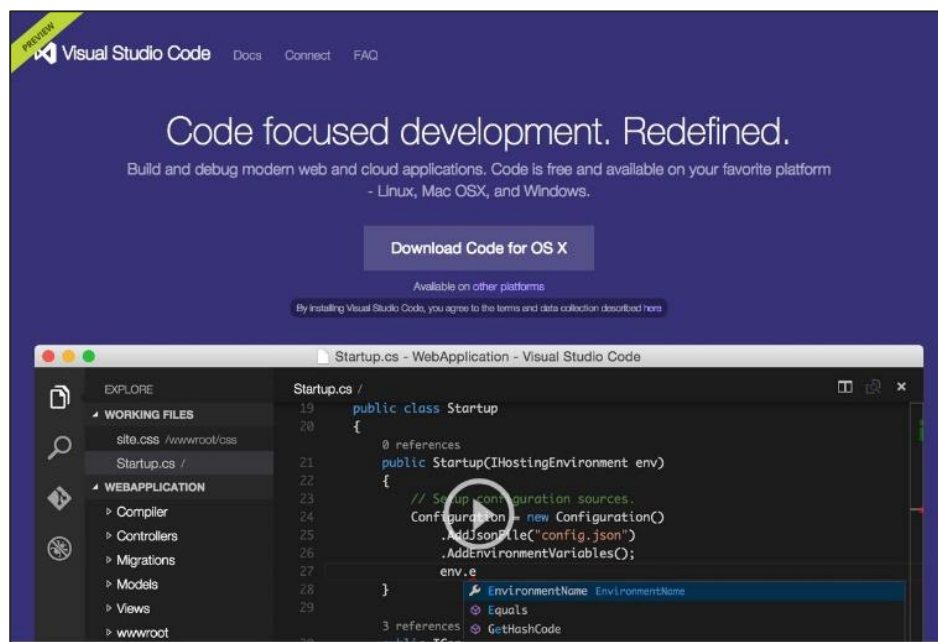
## Visual Studio Code

---

This is open source IDE from Visual Studio. It is available for Mac OS X, Linux, and Windows platforms. VScode is available at <https://code.visualstudio.com>

## Installation on Windows

Download Visual Studio Code for Windows.

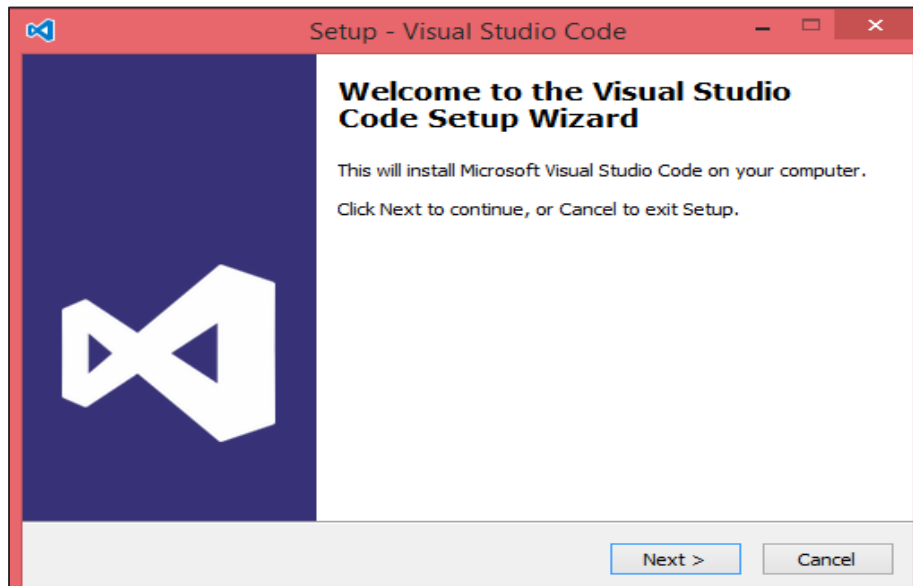


Double-click on VSCodeSetup.exe

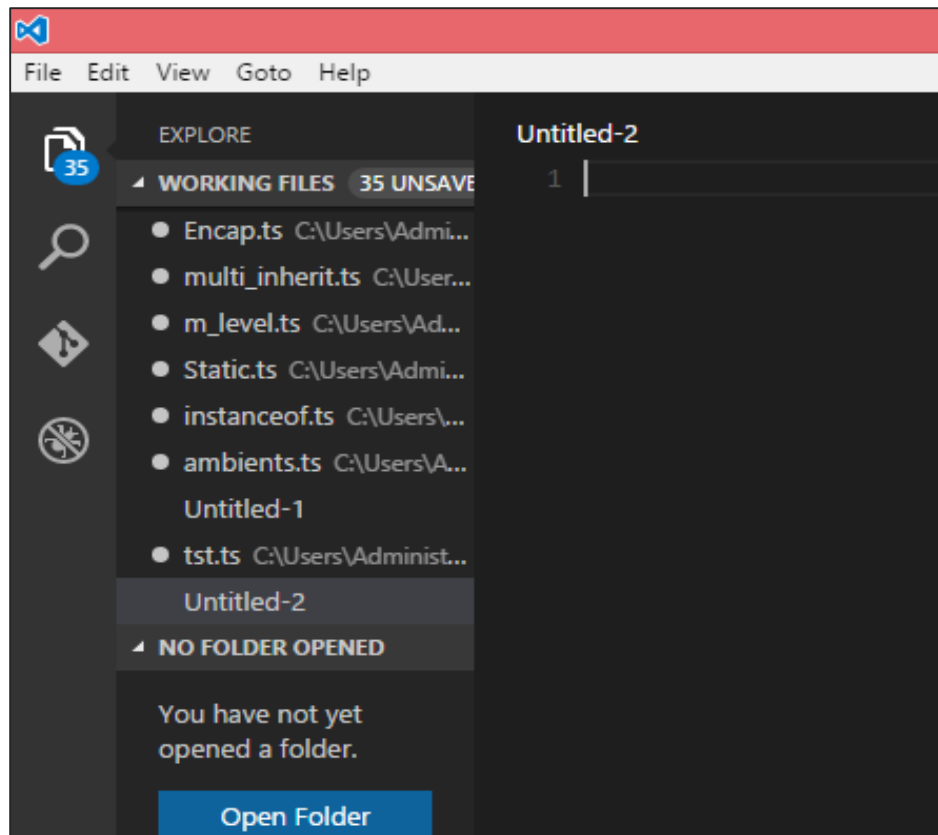


to launch the setup process. This will only take a

minute.

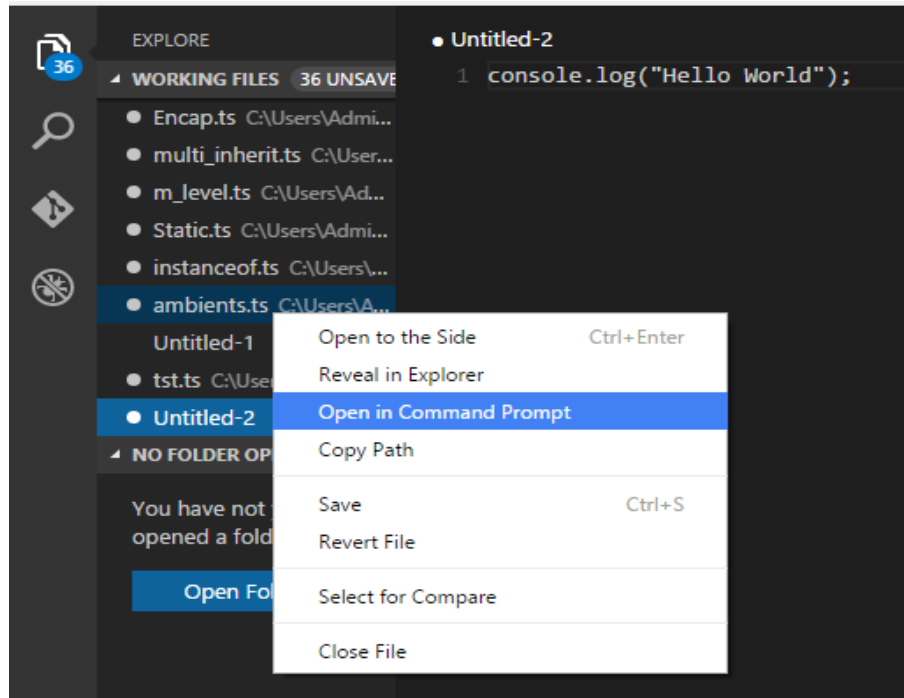


Following is the screenshot of the IDE.



You may directly traverse to the file's path by a right-click on the file -> open in command prompt. Similarly, the **Reveal in Explorer** option shows the file in the File Explorer.





## Installation on Mac OS X

Visual Studio Code's Mac OS X specific installation guide can be found at <https://code.visualstudio.com/Docs/editor/setup>

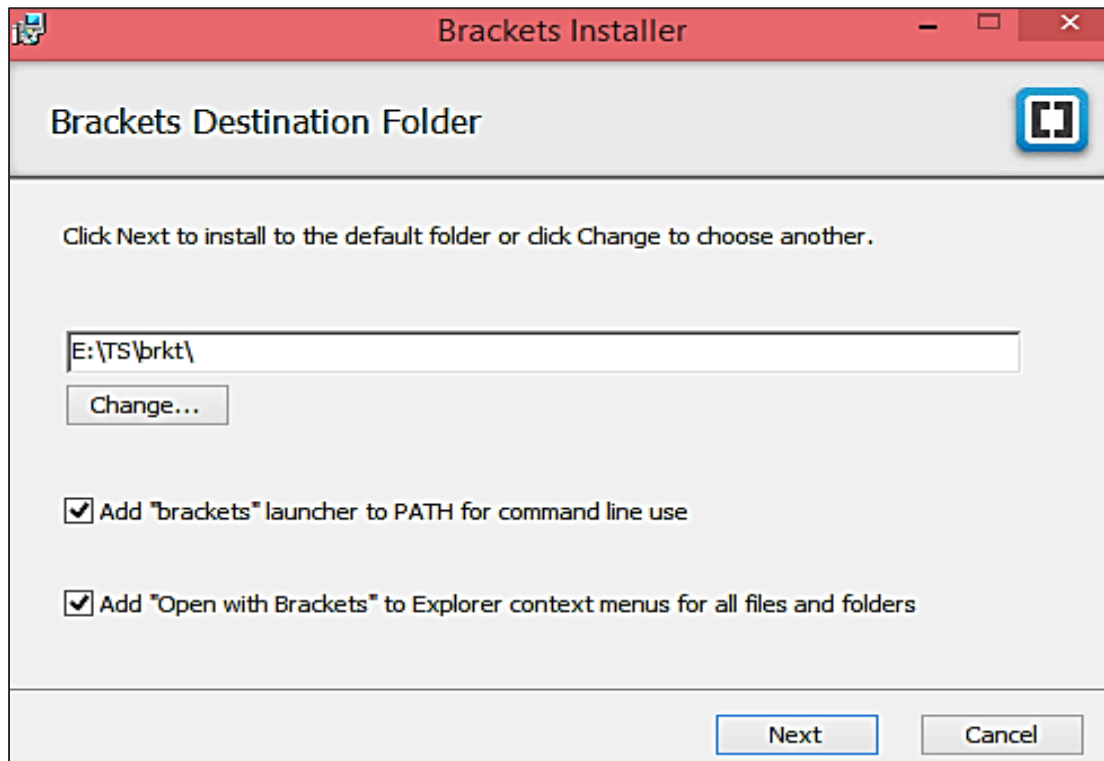
## Installation on Linux

Linux specific installation guide for Visual Studio Code can be found at <https://code.visualstudio.com/Docs/editor/setup>

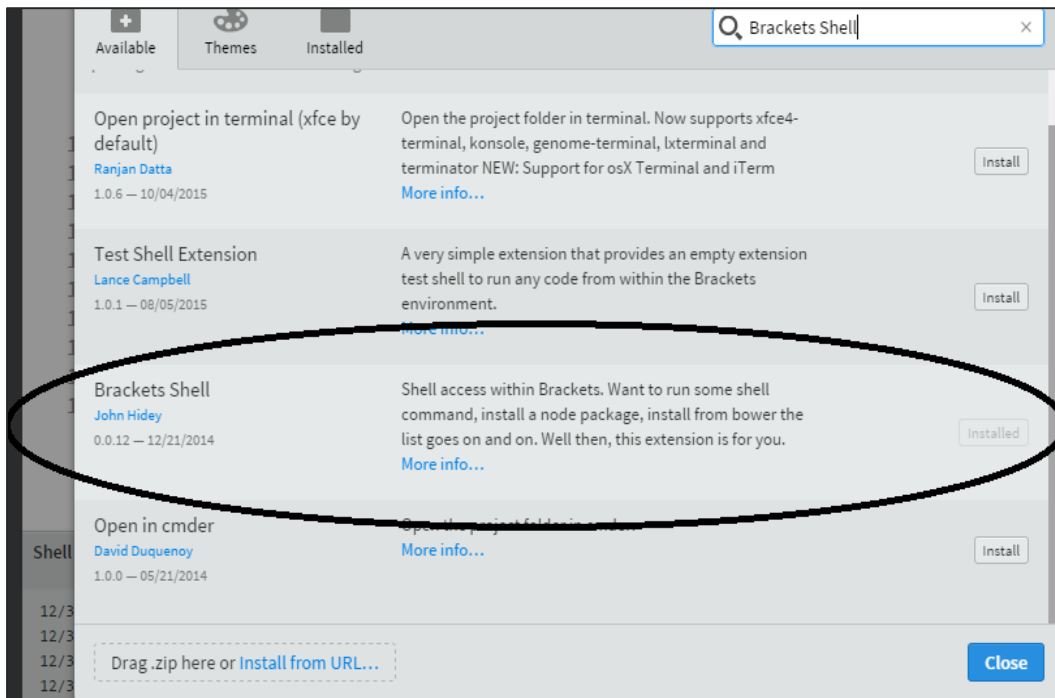
## Brackets


---

Brackets is a free open-source editor for web development, created by Adobe Systems. It is available for Linux, Windows and Mac OS X. Brackets is available at <http://brackets.io>



You can run DOS prompt/Shell within Brackets itself by adding one more extension Brackets Shell.



Upon installation, you will find an icon of shell on the right hand side of the editor . Once you click on the icon, you will see the shell window as shown in the following screenshot.

```

Shell
D:\ts-projects>dir

Volume in drive D is New Volume
Volume Serial Number is B86C-C26C

Directory of D:\ts-projects

    10:23 PM    <DIR>        .
    10:23 PM    <DIR>        ..
           0 File(s)            0 bytes
           2 Dir(s)  93,937,332,224 bytes free

D:\ts-projects>

```

You are all set!!!

### 3. ES6 – Syntax

**Syntax** defines the set of rules for writing programs. Every language specification defines its own syntax.

A JavaScript program can be composed of:

- **Variables:** Represents a named memory block that can store values for the program.
- **Literals:** Represents constant/fixed values.
- **Operators:** Symbols that define how the operands will be processed.
- **Keywords:** Words that have a special meaning in the context of a language.

The following table lists some keywords in JavaScript. Some commonly used keywords are listed in the following table.

break	as	any	Switch
case	if	throw	Else
var	number	string	Get
module	type	instanceof	Typeof
finally	for	enum	Export
while	void	this	New
null	super	Catch	let
static	return	True	False

- **Modules:** Represents code blocks that can be reused across different programs/scripts.
- **Comments:** Used to improve code readability. These are ignored by the JavaScript engine.
- **Identifiers:** These are the names given to elements in a program like variables, functions, etc. The rules for identifiers are:
  - Identifiers can include both, characters and digits. However, the identifier cannot begin with a digit.

- Identifiers cannot include special symbols except for underscore (\_) or a dollar sign (\$).
- Identifiers cannot be keywords. They must be unique.
- Identifiers are case sensitive. Identifiers cannot contain spaces.

The following table illustrates some valid and invalid identifiers.

Examples of valid identifiers	Examples of invalid identifiers
firstName first_name num1 \$result	Var# first name first-name 1number

## Whitespace and Line Breaks

ES6 ignores spaces, tabs, and newlines that appear in programs. You can use spaces, tabs, and newlines freely in your program and you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

## JavaScript is Case-sensitive

JavaScript is case-sensitive. This means that JavaScript differentiates between the uppercase and the lowercase characters.

## Semicolons are Optional

Each line of instruction is called a **statement**. Semicolons are optional in JavaScript.

### Example

```
console.log("hello world")
console.log("We are learning ES6")
```

A single line can contain multiple statements. However, these statements must be separated by a semicolon.

## Comments in JavaScript

---

**Comments** are a way to improve the readability of a program. Comments can be used to include additional information about a program like the author of the code, hints about a function/construct, etc. Comments are ignored by the compiler.

JavaScript supports the following types of comments:

- **Single-line comments (//)**: Any text between a // and the end of a line is treated as a comment.
- **Multi-line comments (/ \* \*/)**: These comments may span multiple lines.

### Example

```
//this is single line comment

/* This is a
Multi-line comment
*/
```

## Your First JavaScript Code

---

Let us start with the traditional "Hello World" example".

```
var message="Hello World"
console.log(message)
```

The program can be analyzed as:

- Line 1 declares a variable by the name message. Variables are a mechanism to store values in a program.
- Line 2 prints the variable's value to the prompt. Here, the console refers to the terminal window. The function log () is used to display the text on the screen.

## Executing the Code

---

We shall use Node.js to execute our code.

**Step 1:** Save the file as Test.js

**Step 2:** Right-click the Test.js file under the working files option in the project-explorer window of the Visual Studio Code.

**Step 3:** Select Open in Command Prompt option.

**Step 4:** Type the following command in Node's terminal window.

```
node Test.js
```

The following output is displayed on successful execution of the file.

```
Hello World
```

## Node.js and JS/ES6

---

ECMAScript 2015(ES6) features are classified into three groups:

- **For Shipping:** These are features that V8 considers stable.
- **Staged Features:** These are almost completed features but not considered stable by the V8 team.
- **In Progress:** These features should be used only for testing purposes.

The first category of features is fully supported and turned on by default by node. Staged features require a runtime - - harmony flag to execute.

A list of component specific CLI flags for Node.js can be found here:  
<https://nodejs.org/api/cli.html>

## The Strict Mode

---

The fifth edition of the ECMAScript specification introduced the Strict Mode. The Strict Mode imposes a layer of constraint on JavaScript. It makes several changes to normal JavaScript semantics.

The code can be transitioned to work in the Strict Mode by including the following:

```
// Whole-script strict mode syntax
"use strict";
v = "Hi! I'm a strict mode script!"; // ERROR: Variable v is not declared
```

In the above snippet, the entire code runs as a constrained variant of JavaScript.

JavaScript also allows to restrict, the Strict Mode within a block's scope as that of a function. This is illustrated as follows:

```
v=15  
function f1()  
{  
  "use strict";  
  var v = "Hi! I'm a strict mode script!";  
}
```

In, the above snippet, any code outside the function will run in the non-script mode. All statements within the function will be executed in the Strict Mode.

## ES6 and Hoisting

---

The JavaScript engine, by default, moves declarations to the top. This feature is termed as **hoisting**. This feature applies to variables and functions. Hoisting allows JavaScript to use a component before it has been declared. However, the concept of hoisting does not apply to scripts that are run in the Strict Mode.

Variable Hoisting and Function Hoisting are explained in the subsequent chapters.



## 4. ES6 – Variables

A **variable**, by definition, is “a named space in the memory” that stores values. In other words, it acts as a container for values in a program. Variable names are called **identifiers**. Following are the naming rules for an identifier:

- Identifiers cannot be keywords.
- Identifiers can contain alphabets and numbers.
- Identifiers cannot contain spaces and special characters, except the underscore (\_) and the dollar (\$) sign.
- Variable names cannot begin with a number.

### Type Syntax

A variable must be declared before it is used. ES5 syntax used the **var** keyword to achieve the same. The ES5 syntax for declaring a variable is as follows.

```
//Declaration using var keyword  
var variable_name
```

ES6 introduces the following variable declaration syntax:

- Using the let
- Using the const

**Variable initialization** refers to the process of storing a value in the variable. A variable may be initialized either at the time of its declaration or at a later point in time.

The traditional ES5 type syntax for declaring and initializing a variable is as follows:

```
//Declaration using var keyword  
var variable_name=value
```

### Example: Using Variables

```
var name="Tom"
```

```
console.log("The value in the variable is: "+name);
```

The above example declares a variable and prints its value.

The following output is displayed on successful execution.

```
The value in the variable is Tom
```

## JavaScript and Dynamic Typing

JavaScript is an un-typed language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold. The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically. This feature is termed as **dynamic typing**.

## JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. Traditionally, JavaScript defines only two scopes-global and local.

- **Global Scope:** A variable with global scope can be accessed from within any part of the JavaScript code.
- **Local Scope:** A variable with a local scope can be accessed from within a function where it is declared.

### Example: Global vs. Local Variable

The following example declares two variables by the name **num** - one outside the function (global scope) and the other within the function (local scope).

```
var num=10
function test()
{
    var num=100
    console.log("value of num in test() "+num)
}
console.log("value of num outside test() "+num)
test()
```

The variable when referred to within the function displays the value of the locally scoped variable. However, the variable **num** when accessed outside the function returns the globally scoped instance.

The following output is displayed on successful execution.

```
value of num outside test() 10  
value of num outside test() 100
```

ES6 defines a new variable scope - The Block scope.

## The Let and Block Scope

The block scope restricts a variable's access to the block in which it is declared. The **var** keyword assigns a function scope to the variable. Unlike the **var** keyword, the **let** keyword allows the script to restrict access to the variable to the nearest enclosing block.

```
"use strict"  
function test()  
{  
    var num=100  
    console.log("value of num in test() "+num)  
    {  
        console.log("Inner Block begins")  
        let num=200  
        console.log("value of num : "+num)  
    }  
}  
test()
```

The script declares a variable **num** within the local scope of a function and re-declares it within a block using the **let** keyword. The value of the locally scoped variable is printed when the variable is accessed outside the inner block, while the block scoped variable is referred to within the inner block.

**Note:** The strict mode is a way to opt in to a restricted variant of JavaScript.

The following output is displayed on successful execution.

```
value of num in test() 100  
Inner Block begins
```

```
value of num : 200
```

### Example: let v/s var

```
var no =10;  
var no =20;  
console.log(no);
```

The following output is displayed on successful execution of the above code.

```
20
```

Let us re-write the same code using the **let** keyword.

```
let no =10;  
let no =20;  
console.log(no);
```

The above code will throw an error: Identifier 'no' has already been declared. Any variable declared using the let keyword is assigned the block scope.

## The const

The **const** declaration creates a read-only reference to a value. It does not mean the value it holds is immutable, just that the variable identifier cannot be reassigned. Constants are block-scoped, much like variables defined using the let statement. The value of a constant cannot change through re-assignment, and it can't be re-declared.

The following rules hold true for a variable declared using the **const** keyword:

- Constants cannot be reassigned a value.
- A constant cannot be re-declared.
- A constant requires an initializer. This means constants must be initialized during its declaration.
- The value assigned to a **const** variable is mutable.

### Example

```
const x=10
```

```
x=12 // will result in an error!!
```

The above code will return an error since constants cannot be reassigned a value. Constants variable are immutable.

## ES6 and Variable Hoisting

The scope of a variable declared with **var** is its current execution context, which is either the enclosing function **or**, for variables declared outside any function, global. Variable hoisting allows the use of a variable in a JavaScript program, even before it is declared.

The following example better explains this concept.

### Example: Variable Hoisting

```
var main = function()
{
    for(var x=0;x<5;x++)
    {
        console.log(x);
    }
    console.log("x can be accessed outside the block scope x value is :"+x);
    console.log('x is hoisted to the function scope');
}
main();
```

The following output is displayed on successful execution of the above code.

```
0 1 2 3 4
x can be accessed outside the block scope x value is :5
x is hoisted to the function scope
```

The JavaScript engine internally represents the script as:

```
var main = function()
{
    var x; // x is hoisted to function scope
    for( x=0;x<5;x++)
    {
```

20

```
        console.log(x);
    }
    console.log("x can be accessed outside the block scope x value is :"+x);
    console.log('x is hoisted to the function scope');
}
main();
```

**Note:** The concept of hoisting applies to variable declaration but not variable initialization. It is recommended to always declare variables at the top of their scope (the top of global code and the top of function code), to enable the code resolve the variable's scope.

## 5. ES6 – Operators

An **expression** is a special kind of statement that evaluates to a value. Every expression is composed of:

- **Operands:** Represents the data.
- **Operator:** Defines how the operands will be processed to produce a value.

Consider the following expression-  $2 + 3$ . Here in the expression, 2 and 3 are operands and the symbol + (plus) is the operator. JavaScript supports the following types of operators:

- Arithmetic operators
- Logical operators
- Relational operators
- Bitwise operators
- Assignment operators
- Ternary/conditional operators
- String operators
- Type operators
- The void operator

### Arithmetic Operators

---

Assume the values in variables **a** and **b** are 10 and 5 respectively.

Operator	Function	Example
+	Addition: Returns the sum of the operands	$a + b$ is 15
-	Subtraction: Returns the difference of the values	$a - b$ is 5
*	Multiplication: Returns the product of the values	$a * b$ is 50

/	Division: Performs a division operation and returns the quotient	a / b is 2
%	Modulus: Performs a division and returns the remainder	a % b is 2
++	Increments the value of the variable by one	a++ is 11
--	Decrements the value of the variable by one	A- - is 9

### Example: Arithmetic Operators

```

var num1=10
var num2=2
var res=0
res= num1+num2
console.log("Sum:      "+ res);
res=num1-num2;
console.log("Difference: "+res)
res=num1*num2
console.log("Product:    "+res)
res=num1/num2
console.log("Quotient:   "+res)
res=num1%num2
console.log("Remainder:  "+res)
num1++
console.log("Value of num1 after increment "+num1)
num2--
console.log("Value of num2 after decrement "+num2)

```

The following output is displayed on successful execution of the above program.

```
Sum: 12
```



Difference: 8  
Product: 20  
Quotient : 5  
Remainder: 0  
Value of num1 after increment: 11  
Value of num2 after decrement: 1

## Relational Operators

Relational operators test or define the kind of relationship between two entities. Relational operators return a boolean value, i.e. true/false.

Assume the value of A is 10 and B is 20.

Operators	Description	Example
>	Greater than	(A > B) is False
<	Lesser than	(A<B) is True
>=	Greater than or equal to	(A >=B) is False
<=	Lesser than or equal to	(A<=B) is True
==	Equality	(A==B) is True
!=	Not equal	(A!=B) is True

### Example

```
var num1 = 5;
var num2 = 9;
console.log("Value of num1: " + num1);
console.log("Value of num2 :" + num2);
var res = num1 > num2;
console.log("num1 greater than num2: " + res);
res = num1 < num2;
console.log("num1 lesser than num2: " + res);
res = num1 >= num2;
console.log("num1 greater than or equal to num2: " + res);
```

```
res = num1 <= num2;
console.log("num1 lesser than or equal to num2: " + res);
res = num1 == num2;
console.log("num1 is equal to num2: " + res);
res = num1 != num2;
console.log("num1 not equal to num2: " + res);
```

The following output is displayed on successful execution of the above code.

```
Value of num1: 5
Value of num2 :9
num1 greater than num2: false
num1 lesser than num2: true
num1 greater than or equal to num2: false
num1 lesser than or equal to num2: true
14 num1 is equal to num2: false
16 num1 not equal to num2: true
```

## Logical Operators

Logical operators are used to combine two or more conditions. Logical operators, too, return a Boolean value. Assume the value of variable A is 10 and B is 20.

Operator	Description	Example
<b>&amp;&amp;</b>	And: The operator returns true only if all the expressions specified return true	(A > 10 && B > 10) is False
<b>  </b>	OR: The operator returns true if at least one of the expressions specified return true	(A > 10    B > 10) is True
<b>!</b>	NOT: The operator returns the inverse of the expression's result. For E.g.: !(7>5) returns false	!(A > 10 ) is True

### Example

```
var avg = 20;
var percentage = 90;
console.log("Value of avg: " + avg + " ,value of percentage: " + percentage);
var res = ((avg > 50) && (percentage > 80));
console.log("(avg>50)&&(percentage>80): ", res);
var res = ((avg > 50) || (percentage > 80));
console.log("(avg>50)||(percentage>80): ", res);
var res = !((avg > 50) && (percentage > 80));
console.log("!((avg>50)&&(percentage>80)): ", res);
```

The following output is displayed on successful execution of the above code.

```
Value of avg: 20 ,value of percentage: 90
(avg>50)&&(percentage>80): false
(avg>50)||(percentage>80): true
!((avg>50)&&(percentage>80)): true
```

## Short-circuit Operators

The **&&** and **||** operators are used to combine expressions.

The **&&** operator returns true only when both the conditions return true. Let us consider an expression:

```
var a=10
var result=( a<10 && a>5)
```

In the above example, `a<10` and `a>5` are two expressions combined by an **&&** operator. Here, the first expression returns false. However, the **&&** operator requires both the expressions to return true. So, the operator skips the second expression.

The **||** operator returns true, if one of the expressions return true. For example:

```
var a=10
var result=( a>5 || a<10)
```

In the above snippet, two expressions `a>5` and `a<10` are combined by a **||** operator. Here, the first expression returns true. Since, the first expression returns true, the **||** operator skips the subsequent expression and returns true.

Due to this behavior of the && and || operator, they are called as short-circuit operators.

## Bitwise Operators

JavaScript supports the following bitwise operators. The following table summarizes JavaScript's bitwise operators.

Operator	Usage	Description
<b>Bitwise AND</b>	<code>a &amp; b</code>	Returns a one in each bit position for which the corresponding bits of both operands are ones
<b>Bitwise OR</b>	<code>a   b</code>	Returns a one in each bit position for which the corresponding bits of either or both operands are ones
<b>Bitwise XOR</b>	<code>a ^ b</code>	Returns a one in each bit position for which the corresponding bits of either but not both operands are ones
<b>Bitwise NOT</b>	<code>~ a</code>	Inverts the bits of its operand
<b>Left shift</b>	<code>a &lt;&lt; b</code>	Shifts a in binary representation b (< 32) bits to the left, shifting in zeroes from the right
<b>Sign-propagating right shift</b>	<code>a &gt;&gt; b</code>	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off
<b>Zero-fill right shift</b>	<code>a &gt;&gt;&gt; b</code>	Shifts a in binary representation b (< 32) bits to the right, discarding bits shifted off, and shifting in zeroes from the left

### Example

```
var a = 2; // Bit presentation 10
var b = 3; // Bit presentation 11
var result;
result = (a & b);
console.log("(a & b) => ", result);
result = (a | b);
console.log("(a | b) => ", result);
```

```
result = (a ^ b);  
console.log("(a ^ b) => ", result);  
result = (~b);  
console.log("(~b) => ", result);  
result = (a << b);  
console.log("(a << b) => ", result);  
result = (a >> b);  
console.log("(a >> b) => ", result);
```

## Output

```
(a & b) => 2  
(a | b) => 3  
(a ^ b) => 1  
(~b) => -4  
(a << b) => 16  
(a >> b) => 0
```

## Assignment Operators

The following table summarizes Assignment operators.

Sr. No.	Operator and Description
1	<b>= (Simple Assignment)</b> Assigns values from the right side operand to the left side operand <b>Example:</b> $C = A + B$ will assign the value of $A + B$ into $C$
2	<b>+= (Add and Assignment)</b> It adds the right operand to the left operand and assigns the result to the left operand. <b>Example:</b> $C += A$ is equivalent to $C = C + A$
3	<b>-= (Subtract and Assignment)</b> It subtracts the right operand from the left operand and assigns the result to the left operand. <b>Example:</b> $C -= A$ is equivalent to $C = C - A$
4	<b>*= (Multiply and Assignment)</b> It multiplies the right operand with the left operand and assigns the result to the left operand. <b>Example:</b> $C *= A$ is equivalent to $C = C * A$
5	<b>/= (Divide and Assignment)</b> It divides the left operand with the right operand and assigns the result to the left operand.

**Note:** The same logic applies to Bitwise operators, so they will become  $<<=$ ,  $>>=$ ,  $>>=$ ,  $\&=$ ,  $|=$  and  $\wedge=$

## Example

```
var a = 12;  
var b = 10;  
a = b;  
console.log("a=b: " + a);  
a += b;  
console.log("a+=b: " + a);  
a -= b;  
console.log("a-=b: " + a);  
a *= b;  
console.log("a*=b: " + a);  
a /= b;  
console.log("a/=b: " + a);  
a %= b;  
console.log("a%=b: " + a);
```

The following output is displayed on successful execution of the above program.

```
a=b: 10  
a+=b: 20  
a-=b: 10  
a*=b: 100  
a/=b: 10  
a%=b: 0
```

## Miscellaneous Operators

Following are some of the miscellaneous operators.

### The negation operator (-)

Changes the sign of a value. The following program is an example of the same.

```
var x=4  
var y=-x;
```

```
console.log("value of x: ",x); //outputs 4
console.log("value of y: ",y); //outputs -4
```

The following output is displayed on successful execution of the above program.

```
value of x: 4
value of y: -4
```

## String Operators: Concatenation operator (+)

The + operator when applied to strings appends the second string to the first. The following program helps to understand this concept.

```
var msg="hello"+"world"
console.log(msg)
```

The following output is displayed on successful execution of the above program.

```
helloworld
```

The concatenation operation doesn't add a space between the strings. Multiple strings can be concatenated in a single statement.

## Conditional Operator (?)

This operator is used to represent a conditional expression. The conditional operator is also sometimes referred to as the ternary operator. Following is the syntax.

```
Test ? expr1 : expr2
```

Where,

**Test:** Refers to the conditional expression

**expr1:** Value returned if the condition is true

**expr2:** Value returned if the condition is false

## Example

```
var num=-2
var result= num > 0 ?"positive":"non-positive"
console.log(result)
```



Line 2 checks whether the value in the variable num is greater than zero. If num is set to a value greater than zero, it returns the string "positive" else a "non-positive" string is returned.

The following output is displayed on successful execution of the above program.

```
non-positive
```

## Type Operators

---

### typeof operator

It is a unary operator. This operator returns the data type of the operand. The following table lists the data types and the values returned by the **typeof** operator in JavaScript.

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"

The following example code displays the number as the output.

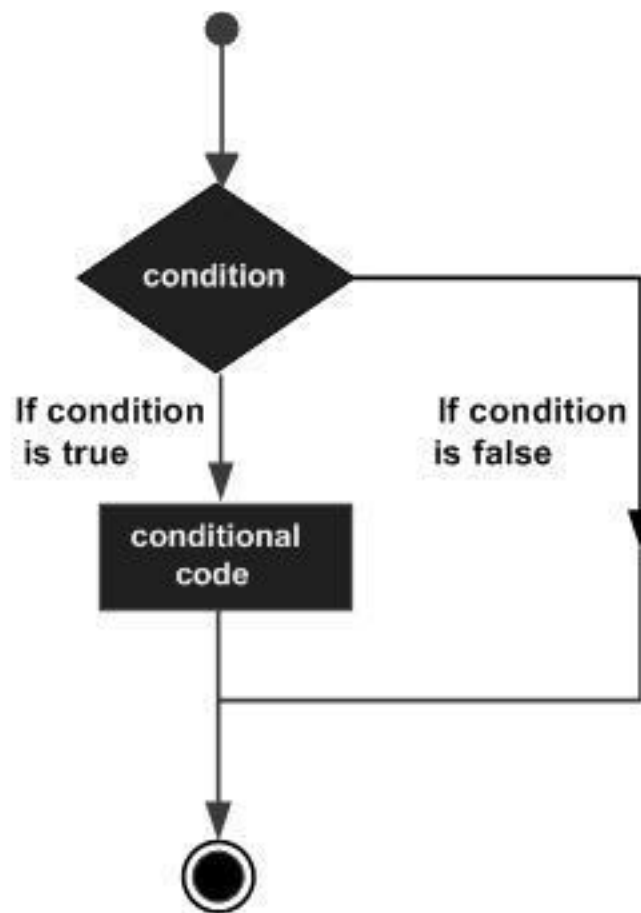
```
var num=12  
console.log(typeof num); //output: number
```

The following output is displayed on successful execution of the above code.

```
number
```

## 6. ES6 – Decision Making

A conditional/decision-making construct evaluates a condition before the instruction/s are executed.



Conditional constructs in JavaScript are classified in the following table.

Statement	Description
<b>if statement</b>	An 'if' statement consists of a Boolean expression followed by one or more statements
<b>if...else statement</b>	An 'if' statement can be followed by an optional 'else' statement, which executes when the Boolean expression is false

<b>The else.. if ladder / nested if statements</b>	You can use one 'if' or 'else if' statement inside another 'if' or 'else if' statement(s)
<b>switch statement</b>	A 'switch' statement allows a variable to be tested for equality against a list of values

## The if Statement

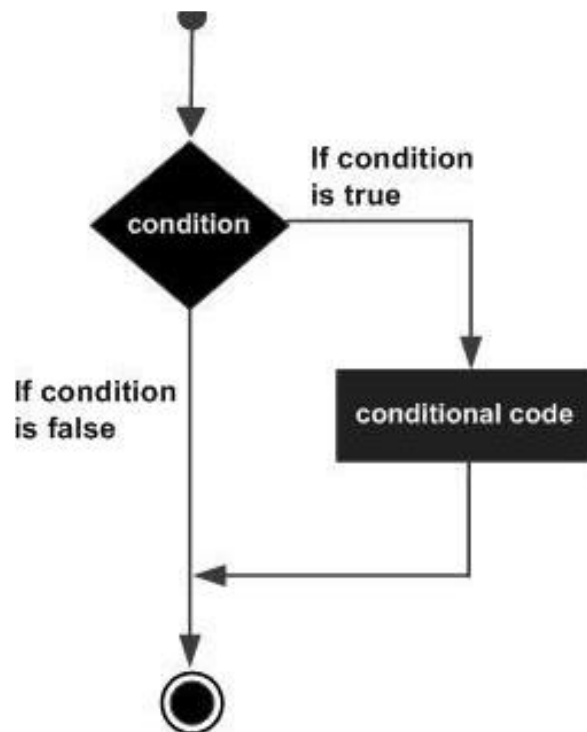
The 'if...else' construct evaluates a condition before a block of code is executed.

Following is the syntax.

```
if(boolean_expression)
{
    // statement(s) will execute if the Boolean expression is true
}
```

If the Boolean expression evaluates to true, then the block of code inside the if statement will be executed. If the Boolean expression evaluates to false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

## Flowchart



## Example

```
var num=5
if (num>0)
{
    console.log("number is positive")
}
```

The following output is displayed on successful execution of the above code.

```
number is positive
```

The above example will print "number is positive" as the condition specified by the if block is true.

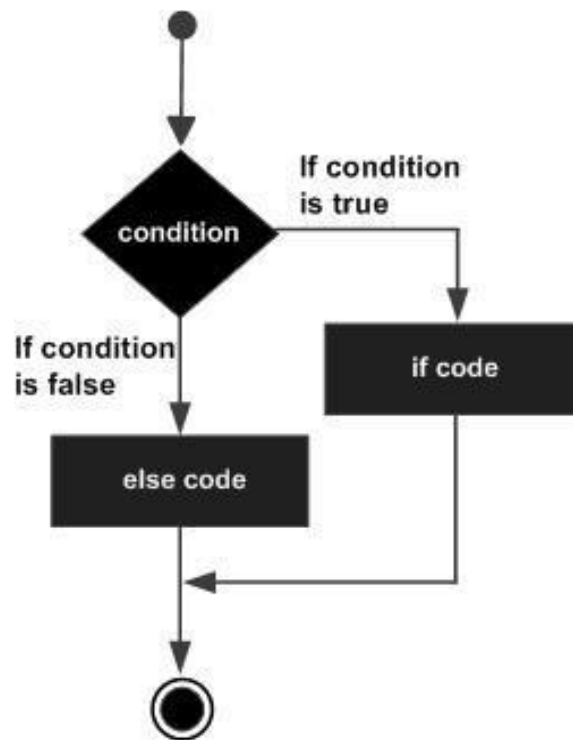
## The if...else Statement

An if can be followed by an optional else block. The else block will execute if the Boolean expression tested by if evaluates to false.

Following is the syntax.

```
if(boolean_expression)
{
    // statement(s) will execute if the Boolean expression is true
}
else
{
    // statement(s) will execute if the Boolean expression is false
}
```

## Flowchart



The if block guards the conditional expression. The block associated with the if statement is executed if the Boolean expression evaluates to true. The if block may be followed by an optional else statement. The instruction block associated with the else block is executed if the expression evaluates to false.

### Example: Simple if...else

```
var num= 12;
if (num % 2==0)
{
    console.log("Even");
}
else
{
    console.log("Odd");
}
```

The above example prints whether the value in a variable is even or odd. The if block checks the divisibility of the value by 2 to determine the same.

The following output is displayed on successful execution of the above code.

Even

## The else...if Ladder

The else...if ladder is useful to test multiple conditions. Following is the syntax of the same.

```
if (boolean_expression1)
{
    //statements if the expression1 evaluates to true
}
else if (boolean_expression2)
{
    //statements if the expression2 evaluates to true
}
else
{
    //statements if both expression1 and expression2 result to false
}
```

When using if...else statements, there are a few points to keep in mind.

- An if can have zero or one else's and it must come after any else if's.
- An if can have zero to many else if's and they must come before the else.
- Once an else if succeeds, none of the remaining else if's or else's will be tested.

### Example: else...if ladder

```
var num=2
if(num > 0)
{
    console.log(num+" is positive")
}
else if(num < 0)
```

```
{
console.log(num+" is negative")
}
else
{
console.log(num+" is neither positive nor negative")
}
```

The code displays whether the value is positive, negative, or zero.

The following output is displayed on successful execution of the above code.

```
2 is positive
```

## The switch...case Statement

The switch statement evaluates an expression, matches the expression's value to a case clause and executes the statements associated with that case.

Following is the syntax.

```
switch(variable_expression)

{
    case constant_expr1:
        {
            //statements;
            break;
        }
    case constant_expr2:
        {
            //statements;
            break;
        }
    default:
```

```
    {  
        //statements;  
        break;  
    }  
}
```

The value of the **variable\_expression** is tested against all cases in the switch. If the variable matches one of the cases, the corresponding code block is executed. If no case expression matches the value of the variable\_expression, the code within the default block is associated.

The following rules apply to a switch statement:

- There can be any number of case statements within a switch.
- The case statements can include only constants. It cannot be a variable or an expression.
- The data type of the variable\_expression and the constant expression must match.
- Unless you put a break after each block of code, the execution flows into the next block.
- The case expression must be unique.
- The default block is optional.



End of ebook preview

If you liked what you saw...

Buy it from our store @ **<https://store.tutorialspoint.com>**

