

Table of Contents

About the Tutorial	
Audience.....	i
Prerequisites.....	i
Copyright and Disclaimer	i
Table of Contents	ii
PART 1: JAVASCRIPT BASICS	1
1. JAVASCRIPT – Overview	2
What is JavaScript?	2
Client-Side JavaScript.....	2
Advantages of JavaScript	3
Limitations of JavaScript.....	3
JavaScript Development Tools.....	3
Where is JavaScript Today?	4
2. JAVASCRIPT – Syntax	5
Your First JavaScript Code	5
Whitespace and Line Breaks.....	6
Semicolons are Optional.....	6
Case Sensitivity	7
Comments in JavaScript	7
3. JAVASCRIPT – Enabling	9
JavaScript in Internet Explorer	9
JavaScript in Firefox.....	9
JavaScript in Chrome	10
JavaScript in Opera	10
Warning for Non-JavaScript Browsers.....	10
4. JAVASCRIPT – Placement	12
JavaScript in <head>...</head> Section.....	12
JavaScript in <body>...</body> Section.....	13
JavaScript in <body> and <head> Sections.....	13
JavaScript in External File	14
5. JAVASCRIPT – Variables	16
JavaScript Datatypes.....	16
JavaScript Variables	16
JavaScript Variable Scope	17
JavaScript Variable Names	18
JavaScript Reserved Words	19
6. JAVASCRIPT – Operators	20
What is an Operator?	20
Arithmetic Operators.....	20
Comparison Operators	23
Logical Operators.....	26

Bitwise Operators	28
Assignment Operators	31
Miscellaneous Operators	34
7. JAVASCRIPT – If-Else	38
Flow Chart of if-else	38
if Statement	39
if...else Statement	40
if...else if... Statement	41
8. JAVASCRIPT – Switch-Case	43
Flow Chart	43
9. JAVASCRIPT – While Loop	47
The while Loop	47
The do...while Loop	49
10. JAVASCRIPT – For Loop	52
The for Loop	52
11. JAVASCRIPT – For-in Loop	55
12. JAVASCRIPT – Loop Control	57
The break Statement	57
The continue Statement	59
Using Labels to Control the Flow	60
13. JAVASCRIPT – Functions	64
Function Definition	64
Calling a Function	65
Function Parameters	66
The return Statement	67
Nested Functions	68
Function () Constructor	70
Function Literals	71
14. JAVASCRIPT – Events	74
What is an Event?	74
onclick Event Type	74
onsubmit Event Type	75
onmouseover and onmouseout	76
HTML 5 Standard Events	77
15. JAVASCRIPT – Cookies	82
What are Cookies?	82
How It Works?	82
Storing Cookies	83
Reading Cookies	84
Setting Cookies Expiry Date	86
Deleting a Cookie	87

16. JAVASCRIPT – Page Redirect	89
What is Page Redirection?.....	89
JavaScript Page Refresh	89
Auto Refresh	89
How Page Re-direction Works?	90
17. JAVASCRIPT – Dialog Box	94
Alert Dialog Box	94
Confirmation Dialog Box.....	95
Prompt Dialog Box	96
18. JAVASCRIPT – Void Keyword	98
19. JAVASCRIPT – Page Printing	101
How to Print a Page?	102
PART 2: JAVASCRIPT OBJECTS	103
20. JAVASCRIPT – Objects	105
Object Properties.....	105
Object Methods.....	105
User-Defined Objects	106
Defining Methods for an Object	108
The ‘with’ Keyword.....	109
21. JAVASCRIPT – Number	112
Number Properties	112
MAX_VALUE	113
MIN_VALUE	114
NaN.....	115
NEGATIVE_INFINITY.....	117
POSITIVE_INFINITY	118
Prototype.....	119
constructor	121
Number Methods	121
toExponential ()	122
toFixed ().....	124
toLocaleString ()	125
toPrecision ()	126
toString ().....	127
valueOf ()	128
22. JAVASCRIPT – Boolean	130
Boolean Properties	130
constructor ().....	130
Prototype.....	131
Boolean Methods	132
toSource ()	133
toString ().....	134
valueOf ()	135

23. JAVASCRIPT – String.....	137
String Properties.....	137
constructor	137
Length.....	138
Prototype.....	139
String Methods	140
charAt().....	142
charCodeAt ().....	143
concat ()	144
indexOf ()	145
lastIndexOf ()	147
localeCompare ()	148
match ()	149
replace ().....	150
Search ().....	153
slice ()	154
split ().....	155
substr ().....	156
substring ().....	157
toLocaleLowerCase()	158
toLocaleUpperCase ()	159
toLowerCase ().....	160
toString ().....	161
toUpperCase ()	162
valueOf ()	163
String HTML Wrappers	164
anchor()	165
big().....	166
blink ().....	167
bold ()	168
fixed ().....	168
fontColor ()	169
fontSize ().....	170
italics ()	171
link ().....	172
small ()	173
strike ().....	174
sub().....	175
sup ().....	176
24. JAVASCRIPT – Arrays.....	178
Array Properties	178
constructor	179
length.....	180
Prototype.....	181
Array Methods.....	182
concat ()	184
every ().....	185
filter ()	187
forEach ()	190

indexOf ()	192
join ()	195
lastIndexOf ()	196
map ()	199
pop ()	201
push ()	202
reduce ()	204
reduceRight ()	207
reverse ()	211
shift ()	212
slice ()	213
some ()	214
sort ()	216
splice ()	217
toString ()	219
unshift ()	220
25. JAVASCRIPT – Date	222
Date Properties	223
constructor	223
Prototype	224
Date Methods	226
Date()	229
getDate()	229
getDay()	230
getFullYear()	231
getHours()	232
getMilliseconds()	233
getMinutes ()	234
getMonth ()	235
getSeconds ()	236
getTime ()	236
getTimezoneOffset ()	237
getUTCDate ()	238
getUTCDay ()	239
getUTCFullYear ()	240
getUTCHours ()	241
getUTCMilliseconds ()	242
getUTCMinutes ()	243
getUTCMonth ()	243
getUTCSeconds ()	244
getYear ()	245
setDate ()	246
setFullYear ()	247
setHours ()	248
setMilliseconds ()	249
setMinutes ()	250
setMonth ()	251
setSeconds ()	252
setTime ()	254

setUTCDate ()	254
setUTCFullYear ()	255
setUTCHours ()	257
setUTCMilliseconds ()	258
setUTCMinutes ()	259
setUTC Month ()	260
setUTCSeconds ()	261
setYear ()	262
toDateString ()	263
toGMTString ()	264
toLocaleDateString ()	265
toLocaleDateString ()	266
toLocaleFormat ()	266
toLocaleString ()	267
toLocaleTimeString ()	268
toSource ()	269
toString ()	270
toTimeString ()	271
toUTCString ()	272
valueOf ()	273
Date Static Methods	274
Date.parse ()	274
Date.UTC ()	275
26. JAVASCRIPT – Math	277
Math Properties	277
Math-E	278
Math-LN2	279
Math-LN10	279
Math-LOG2E	280
Math-LOG10E	281
Math-PI	282
Math-SQRT1_2	283
Math-SQRT2	283
Math Methods	284
abs ()	285
acos ()	287
asin ()	288
atan ()	289
atan2 ()	290
ceil ()	292
cos ()	293
exp ()	295
floor ()	296
log ()	297
max ()	298
min ()	300
pow ()	301
random ()	302
round ()	304

sin ()	305
sqrt ()	306
tan ()	307
toSource ()	309
27. JAVASCRIPT – RegExp	310
Brackets	310
Quantifiers	311
Literal Characters	312
Metacharacters	313
Modifiers	313
RegExp Properties	314
constructor	314
global	315
ignoreCase	316
lastIndex	318
multiline	319
source	320
RegExp Methods	321
exec ()	322
test ()	323
toSource ()	324
toString ()	325
28. JAVASCRIPT – DOM	327
The Legacy DOM	328
The W3C DOM	334
The IE 4 DOM	338
DOM Compatibility	342
PART 3: JAVASCRIPT ADVANCED	344
29. JAVASCRIPT – Errors and Exceptions	345
Syntax Errors	345
Runtime Errors	345
Logical Errors	346
The try...catch...finally Statement	346
The throw Statement	350
The onerror() Method	351
30. JAVASCRIPT – Form Validation	354
Basic Form Validation	356
Data Format Validation	357
31. JAVASCRIPT – Animation	359
Manual Animation	360
Automated Animation	361
Rollover with a Mouse Event	362
32. JAVASCRIPT – Multimedia	365

Checking for Plug-Ins	366
Controlling Multimedia	367
33. JAVASCRIPT – Debugging	369
Error Messages in IE	369
Error Messages in Firefox or Mozilla	370
Error Notifications	371
How to Debug a Script.....	371
Useful Tips for Developers	372
34. JAVASCRIPT – Image Map	374
35. JAVASCRIPT – Browsers	377
Navigator Properties	377
Navigator Methods.....	378
Browser Detection	379

Part 1: JavaScript Basics

1.JAVASCRIPT – OVERVIEW

What is JavaScript?

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

JavaScript was first known as **LiveScript**, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name **LiveScript**. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

The [ECMA-262 Specification](#) defined a standard version of the core JavaScript language.

- JavaScript is a lightweight, interpreted programming language.
- Designed for creating network-centric applications.
- Complementary to and integrated with Java.
- Complementary to and integrated with HTML.
- Open and cross-platform.

Client-Side JavaScript

Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.

It means that a web page need not be a static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.

The JavaScript client-side mechanism provides many advantages over traditional CGI server-side scripts. For example, you might use JavaScript to check if the user has entered a valid e-mail address in a form field.

The JavaScript code is executed when the user submits the form, and only if all the entries are valid, they would be submitted to the Web Server.

JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user initiates explicitly or implicitly.

Advantages of JavaScript

The merits of using JavaScript are:

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language. It lacks the following important features:

- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript cannot be used for networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it

is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Some of them are listed here:

- **Microsoft FrontPage:** Microsoft has developed a popular HTML editor called FrontPage. FrontPage also provides web developers with a number of JavaScript tools to assist in the creation of interactive websites.
- **Macromedia Dreamweaver MX:** Macromedia Dreamweaver MX is a very popular HTML and JavaScript editor in the professional web development crowd. It provides several handy prebuilt JavaScript components, integrates well with databases, and conforms to new standards such as XHTML and XML.
- **Macromedia HomeSite 5:** HomeSite 5 is a well-liked HTML and JavaScript editor from Macromedia that can be used to manage personal websites effectively.

Where is JavaScript Today?

The ECMAScript Edition 5 standard will be the first update to be released in over four years. JavaScript 2.0 conforms to Edition 5 of the ECMAScript standard, and the difference between the two is extremely minor.

The specification for JavaScript 2.0 can be found on the following site:
<http://www.ecmascript.org/>

Today, Netscape's JavaScript and Microsoft's JScript conform to the ECMAScript standard, although both the languages still support the features that are not a part of the standard.

2.JAVASCRIPT – SYNTAX

JavaScript can be implemented using JavaScript statements that are placed within the **<script>... </script>** HTML tags in a web page.

You can place the **<script>** tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the **<head>** tags.

The **<script>** tag alerts the browser program to start interpreting all the text between these tags as a script. A simple syntax of your JavaScript will appear as follows.

```
<script ...>
    JavaScript code
</script>
```

The script tag takes two important attributes:

- **Language:** This attribute specifies what scripting language you are using. Typically, its value will be javascript. Although recent versions of HTML (and XHTML, its successor) have phased out the use of this attribute.
- **Type:** This attribute is what is now recommended to indicate the scripting language in use and its value should be set to "text/javascript".

So your JavaScript syntax will look as follows.

```
<script language="javascript" type="text/javascript">
    JavaScript code
</script>
```

Your First JavaScript Code

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a **"//-->"**. Here **"//"** signifies a comment in JavaScript, so we add that to prevent a browser from reading

the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

This function can be used to write text, HTML, or both. Take a look at the following code.

```
<html>
<body>
<script language="javascript" type="text/javascript">
<!--
    document.write ("Hello World!")
//-->
</script>
</body>
</html>
```

This code will produce the following result:

```
Hello World!
```

Whitespace and Line Breaks

JavaScript ignores spaces, tabs, and newlines that appear in JavaScript 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.

Semicolons are Optional

Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if each of your statements are placed on a separate line. For example, the following code could be written without semicolons.

```
<script language="javascript" type="text/javascript">
```

```
<!--  
    var1 = 10  
    var2 = 20  
//-->  
</script>
```

But when formatted in a single line as follows, you must use semicolons:

```
<script language="javascript" type="text/javascript">  
<!--  
    var1 = 10; var2 = 20;  
//-->  
</script>
```

Note: It is a good programming practice to use semicolons.

Case Sensitivity

JavaScript is a case-sensitive language. This means that the language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.

So the identifiers **Time** and **TIME** will convey different meanings in JavaScript.

NOTE: Care should be taken while writing variable and function names in JavaScript.

Comments in JavaScript

JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a `//` and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters `/*` and `*/` is treated as a comment. This may span multiple lines.

- JavaScript also recognizes the HTML comment opening sequence `<!--`. JavaScript treats this as a single-line comment, just as it does the `//` comment.
- The HTML comment closing sequence `-->` is not recognized by JavaScript so it should be written as `//-->`.

Example

The following example shows how to use comments in JavaScript.

```
<script language="javascript" type="text/javascript">  
<!--  
  
// This is a comment. It is similar to comments in C++  
  
/*  
 * This is a multiline comment in JavaScript  
 * It is very similar to comments in C Programming  
 */  
//-->  
</script>
```


3.JAVASCRIPT – ENABLING

All the modern browsers come with built-in support for JavaScript. Frequently, you may need to enable or disable this support manually. This chapter explains the procedure of enabling and disabling JavaScript support in your browsers: Internet Explorer, Firefox, chrome, and Opera.

JavaScript in Internet Explorer

Here are the steps to turn on or turn off JavaScript in Internet Explorer:

- Follow **Tools -> Internet Options** from the menu.
- Select **Security** tab from the dialog box.
- Click the **Custom Level** button.
- Scroll down till you find the **Scripting** option.
- Select *Enable* radio button under **Active scripting**.
- Finally click OK and come out.

To disable JavaScript support in your Internet Explorer, you need to select **Disable** radio button under **Active scripting**.

JavaScript in Firefox

Here are the steps to turn on or turn off JavaScript in Firefox:

- Open a new tab -> type **about: config** in the address bar.
- Then you will find the warning dialog. Select **I'll be careful, I promise!**
- Then you will find the list of **configure options** in the browser.
- In the search bar, type **javascript.enabled**.
- There you will find the option to enable or disable javascript by right-clicking on the value of that option -> **select toggle**.

If javascript.enabled is true; it converts to false upon clicking **toogle**. If javascript is disabled; it gets enabled upon clicking toggle.

JavaScript in Chrome

Here are the steps to turn on or turn off JavaScript in Chrome:

- Click the Chrome menu at the top right hand corner of your browser.
- Select **Settings**.
- Click **Show advanced settings** at the end of the page.
- Under the **Privacy** section, click the Content settings button.
- In the "Javascript" section, select "Do not allow any site to run JavaScript" or "Allow all sites to run JavaScript (recommended)".

JavaScript in Opera

Here are the steps to turn on or turn off JavaScript in Opera:

- Follow **Tools-> Preferences** from the menu.
- Select **Advanced** option from the dialog box.
- Select **Content** from the listed items.
- Select **Enable JavaScript** checkbox.
- Finally click OK and come out.

To disable JavaScript support in Opera, you should not select the **Enable JavaScript checkbox**.

Warning for Non-JavaScript Browsers

If you have to do something important using JavaScript, then you can display a warning message to the user using **<noscript>** tags.

You can add a **noscript** block immediately after the script block as follows:

```
<html>
<body>

<script language="javascript" type="text/javascript">
<!--
```

```
    document.write ("Hello World!")
//-->
</script>

<noscript>
    Sorry...JavaScript is needed to go ahead.
</noscript>
</body>
</html>
```

Now, if the user's browser does not support JavaScript or JavaScript is not enabled, then the message from </noscript> will be displayed on the screen.

4.JAVASCRIPT – PLACEMENT

There is a flexibility given to include JavaScript code anywhere in an HTML document. However the most preferred ways to include JavaScript in an HTML file are as follows:

- Script in <head>...</head> section.
- Script in <body>...</body> section.
- Script in <body>...</body> and <head>...</head> sections.
- Script in an external file and then include in <head>...</head> section.

In the following section, we will see how we can place JavaScript in an HTML file in different ways.

JavaScript in <head>...</head> Section

If you want to have a script run on some event, such as when a user clicks somewhere, then you will place that script in the head as follows.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
Click here for the result
<input type="button" onclick="sayHello()" value="Say Hello" />
```

```
</body>  
</html>
```

This code will produce the following results:

Click here for the result

Say Hello

JavaScript in <body>...</body> Section

If you need a script to run as the page loads so that the script generates content in the page, then the script goes in the <body> portion of the document. In this case, you would not have any function defined using JavaScript. Take a look at the following code.

```
<html>  
<head>  
</head>  
<body>  
<script type="text/javascript">  
<!--  
document.write("Hello World")  
//-->  
</script>  
<p>This is web page body </p>  
</body>  
</html>
```

This code will produce the following results:

Hello World

This is web page body

JavaScript in <body> and <head> Sections

You can put your JavaScript code in <head> and <body> section altogether as follows.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    alert("Hello World")
}
//-->
</script>
</head>
<body>
<script type="text/javascript">
<!--
document.write("Hello World")
//-->
</script>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>
```

This code will produce the following result.

HelloWorld

Say Hello

JavaScript in External File

As you begin to work more extensively with JavaScript, you will be likely to find that there are cases where you are reusing identical JavaScript code on multiple pages of a site.

You are not restricted to be maintaining identical code in multiple HTML files. The **script** tag provides a mechanism to allow you to store JavaScript in an external file and then include it into your HTML files.

Here is an example to show how you can include an external JavaScript file in your HTML code using **script** tag and its **src** attribute.

```
<html>
<head>
<script type="text/javascript" src="filename.js" ></script>
</head>
<body>
.....
</body>
</html>
```

To use JavaScript from an external file source, you need to write all your JavaScript source code in a simple text file with the extension ".js" and then include that file as shown above.

For example, you can keep the following content in **filename.js** file and then you can use **sayHello** function in your HTML file after including the filename.js file.

```
function sayHello() {
    alert("Hello World")
}
```

5.JAVASCRIPT – VARIABLES

JavaScript Datatypes

One of the most fundamental characteristics of a programming language is the set of data types it supports. These are the type of values that can be represented and manipulated in a programming language.

JavaScript allows you to work with three primitive data types:

- **Numbers**, e.g., 123, 120.50 etc.
- **Strings** of text, e.g. "This text string" etc.
- **Boolean**, e.g. true or false.

JavaScript also defines two trivial data types, **null** and **undefined**, each of which defines only a single value. In addition to these primitive data types, JavaScript supports a composite data type known as **object**. We will cover objects in detail in a separate chapter.

Note: Java does not make a distinction between integer values and floating-point values. All numbers in JavaScript are represented as floating-point values. JavaScript represents numbers using the 64-bit floating-point format defined by the IEEE 754 standard.

JavaScript Variables

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

```
<script type="text/javascript">
<!--
var money;
var name;
```



```
//-->  
</script>
```

You can also declare multiple variables with the same **var** keyword as follows:

```
<script type="text/javascript">  
<!--  
var money, name;  
//-->  
</script>
```

Storing a value in a variable is called **variable initialization**. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

For instance, you might create a variable named **money** and assign the value 2000.50 to it later. For another variable, you can assign a value at the time of initialization as follows.

```
<script type="text/javascript">  
<!--  
var name = "Ali";  
var money;  
money = 2000.50;  
//-->  
</script>
```

Note: Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

JavaScript is **untyped** 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.

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
    var myVar = "local"; // Declare a local variable
    document.write(myVar);
}
//-->
</script>
```

It will produce the following result:

```
Local
```

JavaScript Variable Names

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. These keywords are mentioned in the next section. For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

JavaScript Reserved Words

A list of all the reserved words in JavaScript are given in the following table. They cannot be used as JavaScript variables, functions, methods, loop labels, or any object names.

abstract	else	Instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

6.JAVASCRIPT – OPERATORS

What is an Operator?

Let us take a simple expression **4 + 5 is equal to 9**. Here 4 and 5 are called **operands** and '+' is called the **operator**. JavaScript supports the following types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Let's have a look at all the operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators:

Assume variable A holds 10 and variable B holds 20, then:

S. No.	Operator and Description
1	+ (Addition) Adds two operands Ex: A + B will give 30
2	- (Subtraction) Subtracts the second operand from the first Ex: A - B will give -10

3	* (Multiplication) Multiply both operands Ex: A * B will give 200
4	/ (Division) Divide the numerator by the denominator Ex: B / A will give 2
5	% (Modulus) Outputs the remainder of an integer division Ex: B % A will give 0
6	++ (Increment) Increases an integer value by one Ex: A++ will give 11
7	-- (Decrement) Decreases an integer value by one Ex: A-- will give 9

Note: Addition operator (+) works for Numeric as well as Strings. e.g. "a" + 10 will give "a10".

Example

The following code shows how to use arithmetic operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
```

```
var b = 10;
var c = "Test";
var linebreak = "<br />";

document.write("a + b = ");
result = a + b;
document.write(result);
document.write(linebreak);

document.write("a - b = ");
result = a - b;
document.write(result);
document.write(linebreak);

document.write("a / b = ");
result = a / b;
document.write(result);
document.write(linebreak);

document.write("a % b = ");
result = a % b;
document.write(result);
document.write(linebreak);

document.write("a + b + c = ");
result = a + b + c;
document.write(result);
document.write(linebreak);
```

```

a = a++;
document.write("a++ = ");
result = a++;
document.write(result);
document.write(linebreak);

b = b--;
document.write("b-- = ");
result = b--;
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and then try...</p>
</body>
</html>

```

Output

```

a + b = 43
a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43Test
a++ = 33
b-- = 10

```

Set the variables to different values and then try...

Comparison Operators

JavaScript supports the following comparison operators:

Assume variable A holds 10 and variable B holds 20, then:

S.No	Operator and Description
1	== (Equal) Checks if the value of two operands are equal or not, if yes, then the condition becomes true. Ex: (A == B) is not true.
2	!= (Not Equal) Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true. Ex: (A != B) is true.
3	> (Greater than) Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true. Ex: (A > B) is not true.
4	< (Less than) Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true. Ex: (A < B) is true.
5	>= (Greater than or Equal to) Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true. Ex: (A >= B) is not true.

6	<p><= (Less than or Equal to)</p> <p>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A <= B) is true.</p>
---	---

Example

The following code shows how to use comparison operators in JavaScript.

```

<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";

document.write("(a == b) => ");
result = (a == b);
document.write(result);
document.write(linebreak);

document.write("(a < b) => ");
result = (a < b);
document.write(result);
document.write(linebreak);

document.write("(a > b) => ");
result = (a > b);
document.write(result);
document.write(linebreak);

```

```
document.write("(a != b) => ");  
result = (a != b);  
document.write(result);  
document.write(linebreak);
```

```
document.write("(a >= b) => ");  
result = (a >= b);  
document.write(result);  
document.write(linebreak);
```

```
document.write("(a <= b) => ");  
result = (a <= b);  
document.write(result);  
document.write(linebreak);
```

```
//-->  
</script>
```

```
<p>Set the variables to different values and different operators and then  
try...</p>
```

```
</body>
```

```
</html>
```

Output

```
(a == b) => false  
(a < b) => true  
(a > b) => false  
(a != b) => true  
(a >= b) => false
```

```
(a <= b) => true
```

Set the variables to different values and different operators and then try...

Logical Operators

JavaScript supports the following logical operators:

Assume variable A holds 10 and variable B holds 20, then:

S.No	Operator and Description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	 (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A B) is true.
3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Example

Try the following code to learn how to implement Logical Operators in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = true;
var b = false;
var linebreak = "<br />";

document.write("(a && b) => ");
result = (a && b);
document.write(result);
document.write(linebreak);

document.write("(a || b) => ");
result = (a || b);
document.write(result);
document.write(linebreak);

document.write("!(a && b) => ");
result = (!(a && b));
document.write(result);
document.write(linebreak);

//-->
</script>
```

```
<p>Set the variables to different values and different operators and then  
try...</p>  
</body>  
</html>
```

Output

```
(a && b) => false  
(a || b) => true  
!(a && b) => true
```

Set the variables to different values and different operators and then
try...

Bitwise Operators

JavaScript supports the following bitwise operators:

Assume variable A holds 2 and variable B holds 3, then:

S.No	Operator and Description
1	& (Bitwise AND) It performs a Boolean AND operation on each bit of its integer arguments. Ex: (A & B) is 2.
2	 (BitWise OR) It performs a Boolean OR operation on each bit of its integer arguments. Ex: (A B) is 3.
3	^ (Bitwise XOR)

	<p>It performs a Boolean exclusive OR operation on each bit of its integer arguments. Exclusive OR means that either operand one is true or operand two is true, but not both.</p> <p>Ex: (A ^ B) is 1.</p>
4	<p>~ (Bitwise Not)</p> <p>It is a unary operator and operates by reversing all the bits in the operand.</p> <p>Ex: (~B) is -4.</p>
5	<p><< (Left Shift)</p> <p>It moves all the bits in its first operand to the left by the number of places specified in the second operand. New bits are filled with zeros. Shifting a value left by one position is equivalent to multiplying it by 2, shifting two positions is equivalent to multiplying by 4, and so on.</p> <p>Ex: (A << 1) is 4.</p>
6	<p>>> (Right Shift)</p> <p>Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.</p> <p>Ex: (A >> 1) is 1.</p>
7	<p>>>> (Right shift with Zero)</p> <p>This operator is just like the >> operator, except that the bits shifted in on the left are always zero.</p> <p>Ex: (A >>> 1) is 1.</p>

Example

Try the following code to implement Bitwise operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
```

```
<!--  
var a = 2; // Bit presentation 10  
var b = 3; // Bit presentation 11  
var linebreak = "<br />";  
  
document.write("(a & b) => ");  
result = (a & b);  
document.write(result);  
document.write(linebreak);  
  
document.write("(a | b) => ");  
result = (a | b);  
document.write(result);  
document.write(linebreak);  
  
document.write("(a ^ b) => ");  
result = (a ^ b);  
document.write(result);  
document.write(linebreak);  
  
document.write("(~b) => ");  
result = (~b);  
document.write(result);  
document.write(linebreak);  
  
document.write("(a << b) => ");  
result = (a << b);  
document.write(result);  
document.write(linebreak);
```

```

document.write("(a >> b) => ");
result = (a >> b);
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>

```

Output

```

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

Set the variables to different values and different operators and then
try...

```

Assignment Operators

JavaScript supports the following assignment operators:

S.No	Operator and Description
1	= (Simple Assignment) Assigns values from the right side operand to the left side operand

	Ex: $C = A + B$ will assign the value of $A + B$ into C
2	+= (Add and Assignment) It adds the right operand to the left operand and assigns the result to the left operand. Ex: $C += A$ is equivalent to $C = C + A$
3	-= (Subtract and Assignment) It subtracts the right operand from the left operand and assigns the result to the left operand. Ex: $C -= A$ is equivalent to $C = C - A$
4	*= (Multiply and Assignment) It multiplies the right operand with the left operand and assigns the result to the left operand. Ex: $C *= A$ is equivalent to $C = C * A$
5	/= (Divide and Assignment) It divides the left operand with the right operand and assigns the result to the left operand. Ex: $C /= A$ is equivalent to $C = C / A$
6	%= (Modules and Assignment) It takes modulus using two operands and assigns the result to the left operand. Ex: $C \% = A$ is equivalent to $C = C \% A$

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

Example

Try the following code to implement assignment operator in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 33;
var b = 10;
var linebreak = "<br />";

document.write("Value of a => (a = b) => ");
result = (a = b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a += b) => ");
result = (a += b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a -= b) => ");
result = (a -= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a *= b) => ");
result = (a *= b);
document.write(result);
document.write(linebreak);
```

```
document.write("Value of a => (a /= b) => ");
result = (a /= b);
document.write(result);
document.write(linebreak);

document.write("Value of a => (a %= b) => ");
result = (a %= b);
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

```
Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0

Set the variables to different values and different operators and then
try...
```

Miscellaneous Operators

We will discuss two operators here that are quite useful in JavaScript: the **conditional operator (? :)** and the **typeof operator**.

Conditional Operator (? :)

The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

S.No	Operator and Description
1	? : (Conditional) If Condition is true? Then value X : Otherwise value Y

Example

Try the following code to understand how the Conditional Operator works in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = 20;
var linebreak = "<br />";

document.write ("((a > b) ? 100 : 200) => ");
result = (a > b) ? 100 : 200;
document.write(result);
document.write(linebreak);

document.write ("((a < b) ? 100 : 200) => ");
```

```
result = (a < b) ? 100 : 200;
document.write(result);
document.write(linebreak);

//-->
</script>

<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

```
((a > b) ? 100 : 200) => 200
((a < b) ? 100 : 200) => 100

Set the variables to different values and different operators and then
try...
```

typeof Operator

The **typeof** operator is a unary operator that is placed before its single operand, which can be of any type. Its value is a string indicating the data type of the operand.

The *typeof* operator evaluates to "number", "string", or "boolean" if its operand is a number, string, or boolean value and returns true or false based on the evaluation.

Here is a list of the return values for the **typeof** Operator.

Type	String Returned by typeof
Number	"number"
String	"string"

Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Example

The following code shows how to implement **typeof** operator.

```
<html>
<body>

<script type="text/javascript">
<!--
var a = 10;
var b = "String";
var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");
document.write("Result => ");
document.write(result);
document.write(linebreak);
```

```
//-->
</script>

<p>Set the variables to different values and different operators and then
try...</p>
</body>
</html>
```

Output

```
Result => B is String
Result => A is Numeric

Set the variables to different values and different operators and then
try...
```