



JAVASCRIPT

javascript language

tutorialspoint
SIMPLY EASY LEARNING

www.tutorialspoint.com



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



<https://twitter.com/tutorialspoint>

About the Tutorial

JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform.

Audience

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

Prerequisites

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

Copyright and Disclaimer

© Copyright 2015 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

Table of Contents

About the Tutorial	i
Audience.....	i
Prerequisites.....	i
Copyright and Disclaimer	i
Table of Contents	ii
PART 1: JAVASCRIPT BASICS	1
1. 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. Syntax	5
Your First JavaScript Code	5
Whitespace and Line Breaks.....	6
Semicolons are Optional.....	6
Case Sensitivity	7
Comments in JavaScript	7
3. 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. 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. Variables.....	16
JavaScript Datatypes.....	16
JavaScript Variables	16
JavaScript Variable Scope	17
JavaScript Variable Names	18
JavaScript Reserved Words	19
6. 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. If-Else	38
Flow Chart of if-else	38
if Statement	39
if...else Statement	40
if...else if... Statement	41
8. Switch-Case	43
Flow Chart	43
9. While Loop	47
The while Loop	47
The do...while Loop	49
10. For Loop	52
The for Loop	52
11. For-in Loop	55
12. Loop Control	57
The break Statement	57
The continue Statement	59
Using Labels to Control the Flow	60
13. 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. 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. 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. Page Redirect	89
What is Page Redirection?	89
JavaScript Page Refresh	89
Auto Refresh	89
How Page Re-direction Works?	90
17. Dialog Box.....	94
Alert Dialog Box	94
Confirmation Dialog Box.....	95
Prompt Dialog Box	96
18. Void Keyword	98
19. Page Printing.....	101
How to Print a Page?	102
PART 2: JAVASCRIPT OBJECTS	103
20. Objects.....	105
Object Properties.....	105
Object Methods.....	105
User-Defined Objects	106
Defining Methods for an Object	108
The 'with' Keyword.....	109
21. Number.....	111
Number Properties	111
MAX_VALUE	112
MIN_VALUE	113
NaN.....	114
NEGATIVE_INFINITY.....	116
POSITIVE_INFINITY	117
Prototype.....	118
constructor	120
Number Methods	120
toExponential ()	121
toFixed ().....	123
toLocaleString ()	124
toPrecision ()	125
toString ().....	126
valueOf ()	127
22. Boolean	129
Boolean Properties	129
constructor ().....	129
Prototype.....	130
Boolean Methods	131
toSource ()	132
toString ().....	133
valueOf ()	134

23. String	136
String Properties.....	136
constructor	136
Length.....	137
Prototype.....	138
String Methods	139
charAt().....	141
charCodeAt ().....	142
contact ()	143
indexOf ()	144
lastIndexOf ()	146
localeCompare ()	147
match ()	148
replace ().....	149
Search ().....	152
slice ()	153
split ().....	154
substr ().....	155
substring ().....	156
toLocaleLowerCase()	157
toLocaleUpperCase ()	158
toLowerCase ().....	159
toString ().....	160
toUpperCase ()	161
valueOf ()	162
String HTML Wrappers	163
anchor()	164
big().....	165
blink ().....	166
bold ()	166
fixed ().....	167
fontColor ()	168
fontSize ().....	169
italics ()	170
link ().....	171
small ()	172
strike ().....	173
sub().....	174
sup ().....	175
 24. Arrays	 177
Array Properties	177
constructor	178
length.....	179
Prototype.....	180
Array Methods.....	181
concat ()	183
every ().....	184
filter ()	186
forEach ()	189

indexOf ()	191
join ()	194
lastIndexOf ()	195
map ()	198
pop ()	200
push ()	201
reduce ()	203
reduceRight ()	206
reverse ()	210
shift ()	211
slice ()	212
some ()	213
sort ()	215
splice ()	216
toString ()	218
unshift ()	218
25. Date	220
Date Properties	221
constructor	221
Prototype	222
Date Methods	224
Date()	227
getDate()	227
getDay()	228
getFullYear()	229
getHours()	230
getMilliseconds()	231
getMinutes ()	232
getMonth ()	233
getSeconds ()	234
getTime ()	234
getTimezoneOffset ()	235
getUTCDate ()	236
getUTCDay ()	237
getUTCFullYear ()	238
getUTCHours ()	239
getUTCMilliseconds ()	240
getUTCMinutes ()	241
getUTCMonth ()	241
getUTCSeconds ()	242
getYear ()	243
setDate ()	244
setFullYear ()	245
setHours ()	246
setMilliseconds ()	247
setMinutes ()	248
setMonth ()	249
setSeconds ()	250
setTime ()	252

setUTCDate ()	252
setUTCFullYear ()	253
setUTCHours ()	255
setUTCMilliseconds ()	256
setUTCMinutes ()	257
setUTC Month ()	258
setUTCSeconds ()	259
setYear ()	260
toDateString ()	261
toGMTString ()	262
toLocaleDateString ()	263
toLocaleDateString ()	264
toLocaleFormat ()	264
toLocaleString ()	265
toLocaleTimeSring ()	266
toSource ()	267
toString ()	268
toTimeString ()	269
toUTCString ()	270
valeOf ()	271
Date Static Methods	272
Date.parse ()	272
Date.UTC ()	273
26. Math	275
Math Properties	275
Math-E	276
Math-LN2	277
Math-LN10	277
Math-LOG2E	278
Math-LOG10E	279
Math-PI	280
Math-SQRT1_2	280
Math-SQRT2	281
Math Methods	282
abs ()	283
acos ()	284
asin ()	286
atan ()	287
atan2 ()	288
ceil ()	290
cos ()	291
exp ()	292
floor ()	294
log ()	295
max ()	296
min ()	298
pow ()	299
random ()	300
round ()	301

sin ()	303
sqrt ()	304
tan ()	305
toSource ()	306
27. RegExp	308
Brackets	308
Quantifiers	309
Literal Characters	310
Metacharacters	311
Modifiers	311
RegExp Properties	312
constructor	312
global	313
ignoreCase	314
lastIndex	316
multiline	317
source	318
RegExp Methods	319
exec ()	320
test ()	321
toSource ()	322
toString ()	323
28. DOM	325
The Legacy DOM	326
The W3C DOM	332
The IE 4 DOM	336
DOM Compatibility	340
PART 3: JAVASCRIPT ADVANCED	342
29. Errors and Exceptions	343
Syntax Errors	343
Runtime Errors	343
Logical Errors	344
The try...catch...finally Statement	344
The throw Statement	348
The onerror() Method	349
30. Form Validation	352
Basic Form Validation	354
Data Format Validation	355
31. Animation	357
Manual Animation	358
Automated Animation	359
Rollover with a Mouse Event	360
32. Multimedia	363

Checking for Plug-Ins	364
Controlling Multimedia	365
33. Debugging.....	367
Error Messages in IE	367
Error Messages in Firefox or Mozilla	368
Error Notifications	369
How to Debug a Script.....	369
Useful Tips for Developers	370
34. Image Map.....	372
35. Browsers.....	375
Navigator Properties	375
Navigator Methods.....	376
Browser Detection	377

Part 1: JavaScript Basics

1. 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 web sites.
- **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. 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 your 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 new lines 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. 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'm a developer .
- 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 toggle . 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. 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. 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. 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

	<p>the right operand, if yes , then the condition becomes true.</p> <p>Ex: (A > B) is not true.</p>
4	<p>< (Less than)</p> <p>Checks if the value of the left operand is less than the value of the right operand, if yes , then the condition becomes true.</p> <p>Ex: (A < B) is true.</p>
5	<p>>= (Greater than or Equal to)</p> <p>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.</p> <p>Ex: (A >= B) is not true.</p>
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) 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. Ex: (A ^ B) is 1.
4	~ (Bitwise Not) It is a unary operator and operates by reversing all the bits in the operand.

	Ex: (~B) is -4.
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 B 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	<p>$\% =$ (Modules and Assignment)</p> <p>It takes modulus using two operands and assigns the result to the left operand.</p> <p>Ex: $C \% = A$ is equivalent to $C = C \% A$</p>

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

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	<code>?:</code> (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...

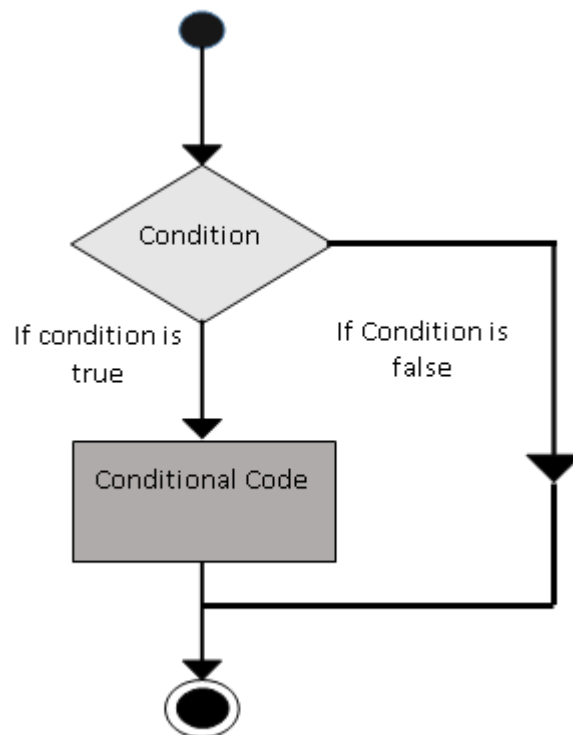
7. IF-ELSE

While writing a program, there may be a situation when you need to adopt one out of a given set of paths. In such cases, you need to use conditional statements that allow your program to make correct decisions and perform right actions.

JavaScript supports conditional statements which are used to perform different actions based on different conditions. Here we will explain the if..else statement.

Flow Chart of if-else

The following flow chart shows how the if-else statement works.



JavaScript supports the following forms of if..else statement:

- if statement
- if...else statement
- if...else if... statement

if Statement

The if statement is the fundamental control statement that allows JavaScript to make decisions and execute statements conditionally.

Syntax

The syntax for a basic if statement is as follows:

```
if (expression){  
    Statement(s) to be executed if expression is true  
}
```

Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

Example

Try the following example to understand how the if statement works.

```
<html>  
<body>  
  
<script type="text/javascript">  
  <!--  
  var age = 20;  
  if( age > 18 ){  
    document.write("<b>Qualifies for driving</b>");  
  }  
  //-->  
</script>  
  
<p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

Qualifies for driving
Set the variable to different value and then try...

if...else Statement

The if...else statement is the next form of control statement that allows JavaScript to execute statements in a more controlled way.

Syntax

The syntax of an if-else statement is as follows:

```
if (expression){
    Statement(s) to be executed if expression is true
}else{
    Statement(s) to be executed if expression is false
}
```

Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the if block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

Example

Try the following code to learn how to implement an if-else statement in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var age = 15;

if( age > 18 ){
    document.write("<b>Qualifies for driving</b>");
}else{
    document.write("<b>Does not qualify for driving</b>");
}
```

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

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Does not qualify for driving
Set the variable to different value and then try...
```

if...else if... Statement

The if...else if... Statement is an advanced form of if...else statement that allows JavaScript to make a correct decision out of several conditions.

Syntax

The syntax of an if-else-if statement is as follows:

```
if (expression 1){
    Statement(s) to be executed if expression 1 is true
}else if (expression 2){
    Statement(s) to be executed if expression 2 is true
}else if (expression 3){
    Statement(s) to be executed if expression 3 is true
}else{
    Statement(s) to be executed if no expression is true
}
```

There is nothing special about this code. It is just a series of if statements, where each if is a part of the else clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the else block is executed.

Example

Try the following code to learn how to implement an if-else-if statement in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var book = "maths";
if( book == "history" ){
    document.write("<b>History Book</b>");
}else if( book == "maths" ){
    document.write("<b>Maths Book</b>");
}else if( book == "economics" ){
    document.write("<b>Economics Book</b>");
}else{
    document.write("<b>Unknown Book</b>");
}
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

Maths Book

Set the variable to different value and then try...

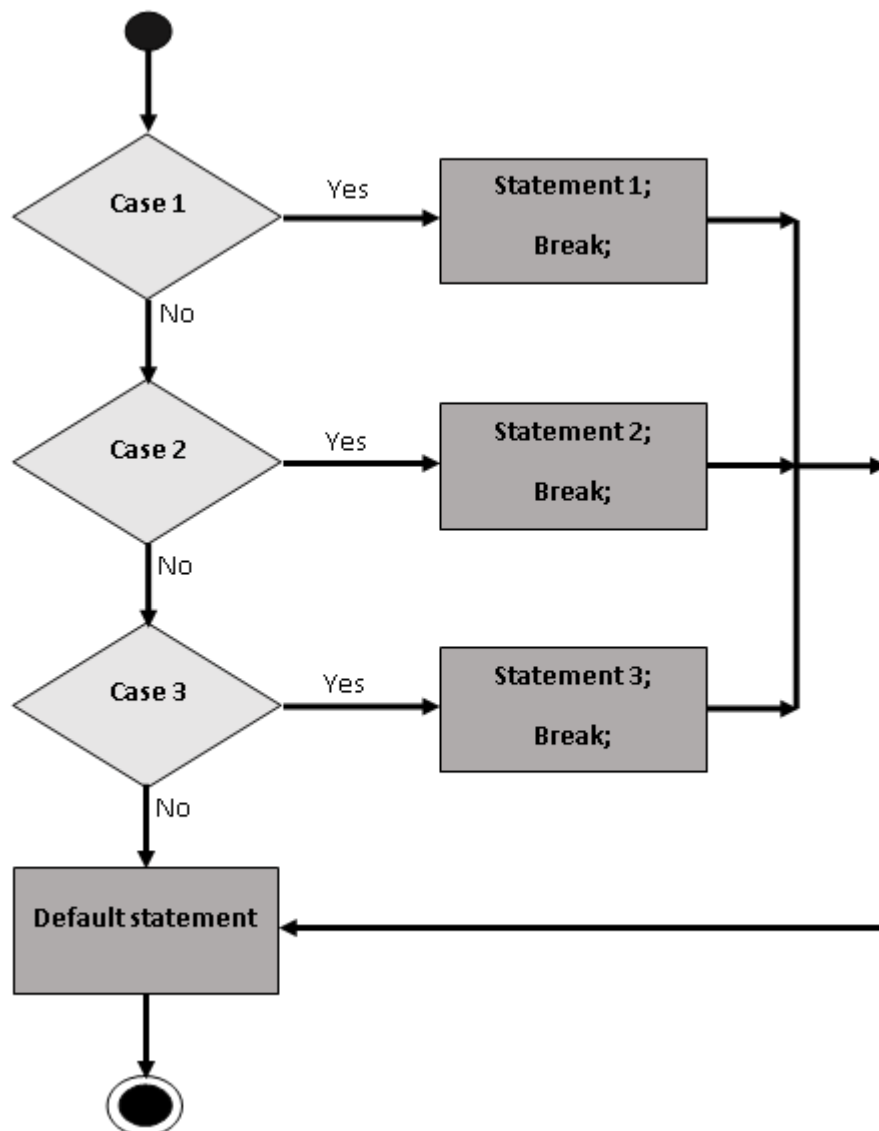
8. SWITCH-CASE

You can use multiple if...else Æ if statements, as in the previous chapter, to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.

Starting with JavaScript 1.2, you can use a switch statement which handles exactly this situation, and it does so more efficiently than repeated if...else if statements.

Flow Chart

The following flow chart explains a switch-case statement works.



Syntax

The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

```
switch (expression)
{
    case condition 1: statement(s)
                        break;
    case condition 2: statement(s)
                        break;
    ...
    case condition n: statement(s)
                        break;
    default: statement(s)
}

```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

We will explain break statement in Loop Control chapter.

Example

Try the following example to implement switch -case statement.

```
<html>
<body>

<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");
switch (grade)
{
    case 'A': document.write("Good job<br />");

```

```

        break;
    case 'B': document.write("Pretty good<br />");
        break;
    case 'C': document.write("Passed<br />");
        break;
    case 'D': document.write("Not so good<br />");
        break;
    case 'F': document.write("Failed<br />");
        break;
    default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>

```

Output

```

Entering switch block
Good job
Exiting switch block
Set the variable to different value and then try...

```

Break statements play a major role in switch-case statements. Try the following code that uses switch-case statement without any break statement.

```

<html>
<body>

<script type="text/javascript">
<!--
var grade='A';
document.write("Entering switch block<br />");

```

```
switch (grade)
{
  case 'A': document.write("Good job<br />");
  case 'B': document.write("Pretty good<br />");
  case 'C': document.write("Passed<br />");
  case 'D': document.write("Not so good<br />");
  case 'F': document.write("Failed<br />");
  default: document.write("Unknown grade<br />")
}
document.write("Exiting switch block");
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade
Exiting switch block
Set the variable to different value and then try...
```


9. WHILE LOOP

While writing a program, you may encounter a situation where you need to perform an action over and over again. In such situations, you would need to write loop statements to reduce the number of lines.

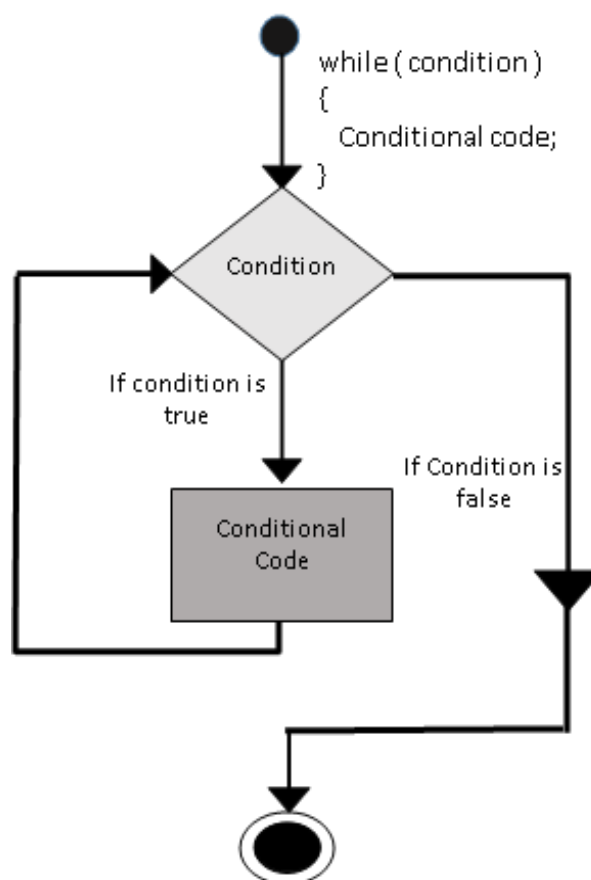
JavaScript supports all the necessary loops to ease down the pressure of programming.

The while Loop

The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

Flow Chart

The flow chart of while loop looks as follows:



Syntax

The syntax of while loop in JavaScript is as follows:

```
while (expression){  
    Statement(s) to be executed if expression is true  
}
```

Example

Try the following example to implement while loop.

```
<html>  
<body>  
  
<script type="text/javascript">  
<!--  
var count = 0;  
document.write("Starting Loop ");  
while (count < 10){  
    document.write("Current Count : " + count + "<br />");  
    count++;  
}  
document.write("Loop stopped!");  
//-->  
</script>  
  
<p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

```
Starting Loop Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4
```

```
Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

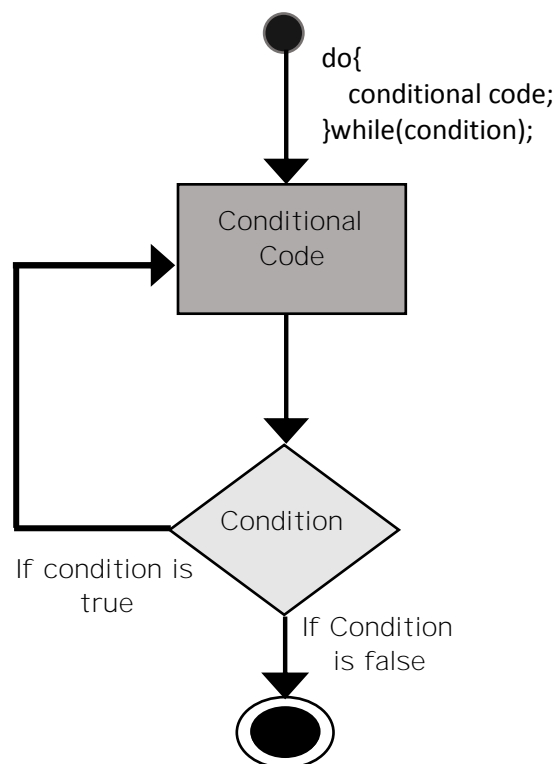
Set the variable to different value and then try...
```

The do...while Loop

The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false.

Flow Chart

The flow chart of a do-while loop would be as follows:



Syntax

The syntax for do-while loop in JavaScript is as follows:

```
do{
    Statement(s) to be executed;
} while (expression);
```

Note: The semicolon is used at the end of the do...while loop.

Example

Try the following example to learn how to implement a do-while loop in JavaScript.

```
<html>
<body>

<script type="text/javascript">
<!--
var count = 0;
document.write("Starting Loop" + "<br />");
do{
    document.write("Current Count : " + count + "<br />");
    count++;
}while (count < 5);
document.write ("Loop stopped!");
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Starting Loop
Current Count : 0
Current Count : 1
Current Count : 2
Current Count : 3
Current Count : 4
Loop Stopped!
```

Set the variable to different value and then try...

10. FOR LOOP

The for Loop

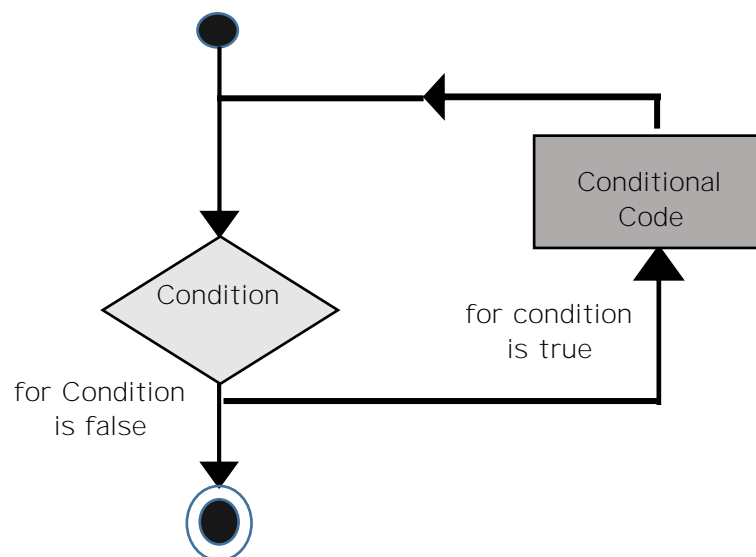
The for loop is the most compact form of looping. It includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter.

You can put all the three parts in a single line separated by semicolons.

Flow Chart

The flow chart of a for loop in JavaScript would be as follows:



Syntax

The syntax of for loop in JavaScript is as follows:

```
for (initialization; test condition; iteration statement){  
    Statement(s) to be executed if test condition is true  
}
```

Example

Try the following example to learn how a for loop works in JavaScript .

```
<html>  
<body>  
  
<script type="text/javascript">  
<!--  
var count;  
document.write("Starting Loop" + "<br />");  
for(count = 0; count < 10; count++){  
    document.write("Current Count : " + count );  
    document.write("<br />");  
}  
document.write("Loop stopped!");  
//-->  
</script>  
  
<p>Set the variable to different value and then try...</p>  
</body>  
</html>
```

Output

```
Starting Loop  
Current Count : 0  
Current Count : 1  
Current Count : 2  
Current Count : 3  
Current Count : 4  
Current Count : 5
```

```
Current Count : 6  
Current Count : 7  
Current Count : 8  
Current Count : 9  
Loop stopped!  
Set the variable to different value and then try...
```


11. FOR-IN LOOP

The for...in loop is used to loop through an object's properties. As we have not discussed Objects yet, you may not feel comfortable with this loop. But once you understand how objects behave in JavaScript, you will find this loop very useful.

Syntax

The syntax of for...in is as follows:

```
for (variablename in object){  
    statement or block to execute  
}
```

In each iteration, one property from object is assigned to variablename and this loop continues till all the properties of the object are exhausted.

Example

Try the following example to see how the for...in loop works with the Navigator object.

```
<html>  
<body>  
  
<script type="text/javascript">  
<!--  
var aProperty;  
document.write("Navigator Object Properties<br /> ");  
for (aProperty in navigator)  
{  
    document.write(aProperty);  
    document.write("<br />");  
}  
document.write ("Exiting from the loop!");  
//-->  
</script>
```

```
<p>Set the variable to different object and then try...</p>  
</body>  
</html>
```

Output

```
Navigator Object Properties  
serviceWorker  
webkitPersistentStorage  
webkitTemporaryStorage  
geolocation  
doNotTrack  
onLine  
languages  
language  
userAgent  
product  
platform  
appVersion  
appName  
appCodeName  
hardwareConcurrency  
maxTouchPoints  
vendorSub  
vendor  
productSub  
cookieEnabled  
mimeType  
plugins  
javaEnabled  
getStorageUpdates  
getGamepads  
webkitGetUserMedia  
vibrate  
getBattery  
sendBeacon  
registerProtocolHandler  
unregisterProtocolHandler  
Exiting from the loop!  
Set the variable to different object and then try...
```

12. LOOP CONTROL

JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop.

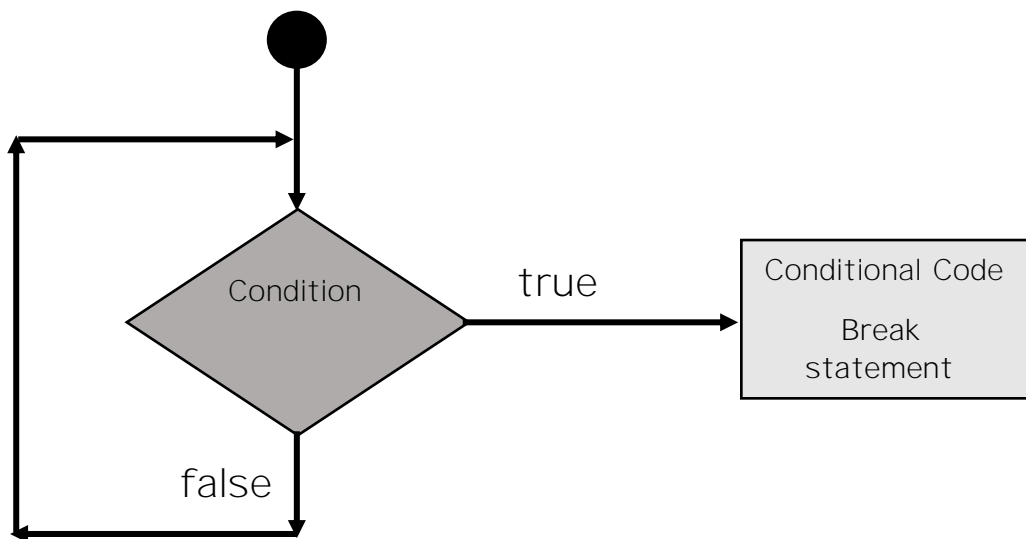
To handle all such situations, JavaScript provides `break` and `continue` statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

The break Statement

The `break` statement, which was briefly introduced with the `switch` statement, is used to exit a loop early, breaking out of the enclosing curly braces.

Flow Chart

The flow chart of a `break` statement would look as follows:



Example

The following example illustrates the use of a `break` statement with a `while` loop. Notice how the loop breaks out early once `x` reaches 5 and reaches to `document.write (..)` statement just below to the closing curly brace:

```
<html>
<body>

<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 20)
{
    if (x == 5){
        break; // breaks out of loop completely
    }
    x = x + 1;
    document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop
2
3
4
5
Exiting the loop!

Set the variable to different value and then try...
```

We have already seen the usage of `break` statement inside a `switch` statement.

The continue Statement

The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

This example illustrates the use of a continue statement with a while loop. Notice how the continue statement is used to skip printing when the index held in variable x reaches 5.

```
<html>
<body>
<script type="text/javascript">
<!--
var x = 1;
document.write("Entering the loop<br /> ");
while (x < 10)
{
    x = x + 1;
    if (x == 5){
        continue; // skip rest of the loop body
    }
    document.write( x + "<br />");
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

<p>Set the variable to different value and then try...</p>
</body>
</html>
```

Output

```
Entering the loop
```

```
2
```

```
3
```

```
4
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
Exiting the loop!
```

Using Labels to Control the Flow

Starting from JavaScript 1.2, a label can be used with `break` and `continue` to control the flow more precisely. A label is simply an identifier followed by a colon (:) that is applied to a statement or a block of code. We will see two different examples to understand how to use labels with `break` and `continue`.

Note: Line breaks are not allowed between the `continue` or `break` statement and its label name. Also, there should not be any other statement in between a label name and associated loop.

Try the following two examples for a better understanding of Labels.

Example 1

The following example shows how to implement Label with a `break` statement.

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop: // This is the label name
for (var i = 0; i < 5; i++)
{
    document.write("Outerloop: " + i + "<br />");
    innerloop:
    for (var j = 0; j < 5; j++)
```

```

{
    if (j > 3 ) break ;           // Quit the innermost loop
    if (i == 2) break innerloop; // Do the same thing
    if (i == 4) break outerloop; // Quit the outer loop
    document.write("Innerloop: " + j + " <br />");
}
}
document.write("Exiting the loop!<br /> ");
//-->
</script>
</body>
</html>

```

Output

```

Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 2
Outerloop: 3
Innerloop: 0
Innerloop: 1
Innerloop: 2
Innerloop: 3
Outerloop: 4
Exiting the loop!

```

Example 2

The following example shows how to implement Label with continue.

```
<html>
```

```
<body>
<script type="text/javascript">
<!--
document.write("Entering the loop!<br /> ");
outerloop:  // This is the label name
for (var i = 0; i < 3; i++)
{
    document.write("Outerloop: " + i + "<br />");
    for (var j = 0; j < 5; j++)
    {
        if (j == 3){
            continue outerloop;
        }
        document.write("Innerloop: " + j + "<br />");
    }
}
document.write("Exiting the loop!<br /> ");
//-->
</script>

</body>
</html>
```

Output

```
Entering the loop!
Outerloop: 0
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 1
Innerloop: 0
Innerloop: 1
Innerloop: 2
Outerloop: 2
Innerloop: 0
Innerloop: 1
```



```
Innerloop: 2  
Exiting the loop!
```

13. FUNCTIONS

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like `alert()` and `write()` in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

JavaScript allows us to write our own functions as well. This section explains how to write your own functions in JavaScript.

Function Definition

Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the `function` keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

The basic syntax is shown here .

```
<script type="text/javascript">
<!--
function functionname(parameter-list)
{
    statements
}
//-->
</script>
```

Example

Try the following example . It defines a function called `sayHello` that takes no parameters :

```
<script type="text/javascript">
<!--
function sayHello()
{
    alert("Hello there");
}
//-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```
<html>
<head>
<script type="text/javascript">
function sayHello()
{
    document.write ("Hello there!");
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello()" value="Say Hello">
</form>

<p>Use different text in write method and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Say Hello

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

Try the following example. We have modified our sayHello function here. Now it takes two parameters.

```
<html>
<head>
<script type="text/javascript">
function sayHello(name, age)
{
    document.write (name + " is " + age + " years old.");
}
</script>
</head>

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Say Hello

Use different parameters inside the function and then try...

The return Statement

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

For example , you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example

Try the following example . It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program .

```
<html>
<head>
<script type="text/javascript">
function concatenate(first, last)
{
    var full;

    full = first + last;
    return full;
}
function secondFunction()
{
    var result;
    result = concatenate('Zara', 'Ali');
    document.write (result );
}
</script>
</head>
```

```

<body>
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>

```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

There is a lot to learn about JavaScript functions , however we have covered the most important concepts in this tutorial.

Nested Functions

Prior to JavaScript 1.2, function definition was allowed only in top level global code, but JavaScript 1.2 allows function definitions to be nested within other functions as well. Still there is a restriction that function definitions may not appear within loops or conditionals. These restrictions on function definitions apply only to function declarations with the `function` statement.

As we'll discuss later in the next chapter, function literals (another feature introduced in JavaScript 1.2) may appear within any JavaScript expression, which means that they can appear within `if` and other statements.

Example

Try the following example to learn how to implement nested functions.

```
<html>
```

```
<head>
<script type="text/javascript">
<!--
function hypotenuse(a, b) {
    function square(x) { return x*x; }

    return Math.sqrt(square(a) + square(b));
}
function secondFunction(){
    var result;
    result = hypotenuse(1,2);
    document.write ( result );
}
//-->
</script>
</head>

<body>
<p>Click the following button to call the function</p>

<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

Function () Constructor

The *function* statement is not the only way to define a new function; you can define your function dynamically using `Function ()` constructor along with the `new` operator.

Note: Constructor is a terminology from Object Oriented Programming. You may not feel comfortable for the first time, which is OK.

Syntax

Following is the syntax to create a function using `Function ()` constructor along with the `new` operator.

```
<script type="text/javascript">
<!--
var variablename = new Function(Arg1, Arg2..., "Function Body");
//-->
</script>
```

The `Function()` constructor expects any number of string arguments. The last argument is the body of the function. It can contain arbitrary JavaScript statements, separated from each other by semicolons.

Notice that the `Function ()` constructor is not passed any argument that specifies a name for the function it creates. The unnamed functions created with the `Function()` constructor are called anonymous functions.

Example

Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
var func = new Function("x", "y", "return x*y;");

function secondFunction(){
    var result;
    result = func(10,20);
```



```

    document.write ( result );
}
//-->
</script>
</head>

<body>
<p>Click the following button to call the function</p>

<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>

<p>Use different parameters inside the function and then try...</p>
</body>
</html>

```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

Function Literals

JavaScript 1.2 introduces the concept of function literals which is another new way of defining functions. A function literal is an expression that defines an unnamed function.

Syntax

The syntax for a function literal is much like a function statement, except that it is used as an expression rather than a statement and no function name is required.

```

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

```

```
var variablename = function(Argument List){
    Function Body
};
//-->
</script>
```

Syntactically, you can specify a function name while creating a literal function as follows.

```
<script type="text/javascript">
<!--
var variablename = function FunctionName(Argument List){
    Function Body
};
//-->
</script>
```

But this name does not have any significance, so it is not worthwhile.

Example

Try the following example . It shows the usage of function literals.

```
<html>
<head>
<script type="text/javascript">
<!--
var func = function(x,y){ return x*y };

function secondFunction(){
    var result;
    result = func(10,20);
    document.write ( result );
}
//-->
</script>
</head>
<body>
```

```
<p>Click the following button to call the function</p>
<form>
<input type="button" onclick="secondFunction()" value="Call Function">
</form>
<p>Use different parameters inside the function and then try...</p>
</body>
</html>
```

Output

Click the following button to call the function

Call Function

Use different parameters inside the function and then try...

14. EVENTS

What is an Event?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](#). Here we will see a few examples to understand the relation between Event and JavaScript.

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

```
<html>
<head>
<script type="text/javascript">
<!--
function sayHello() {
    document.write ("Hello World")
}
//-->
</script>
```

```

</head>
<body>
<p> Click the following button and see result</p>
<input type="button" onclick="sayHello()" value="Say Hello" />
</body>
</html>

```

Output

Click the following button and see result

Say Hello

onsubmit Event Type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a validate() function before submitting a form data to the webserver. If validate() function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```

<html>
<head>
<script type="text/javascript">
<!--
function validation() {
    all validation goes here
    .....
    return either true or false
}
//-->
</script>
</head>

```

```
<body>
<form method="POST" action="t.cgi" onsubmit="return validate()">
.....
<input type="submit" value="Submit" />
</form>
</body>
</html>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The onmouseover event triggers when you bring your mouse over any element and the onmouseout triggers **when you move**

Return Value

Returns the day of the month in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCDate Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getUTCDate() : " + dt.getUTCDate() );
</script>
</body>
</html>
```

Output

```
getUTCDate() : 25
```

getUTCDay ()

JavaScript date getUTCDay() method returns the day of the week in the specified date according to universal time. The value returned by getUTCDay is an integer corresponding to the day of the week: 0 for Sunday, 1 for Monday, 2 for Tuesday, and so on.

Syntax

Its syntax is as follows:

```
Date.getUTCDay ()
```

Return Value

Returns the day of the week in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCDay Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getUTCDay() : " + dt.getUTCDay() );
</script>
</body>
</html>
```

Output

```
getUTCDay() : 1
```

getUTCFullYear ()

Javascript date getUTCFullYear() method returns the year in the specified date according to universal time. The value returned by getUTCFullYear is an absolute number that is compliant with year -2000, for example, 2008.

Syntax

Its syntax is as follows:

```
Date.getUTCFullYear ()
```

Return Value

Returns the year in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCFullYear Method</title>
</head>
```



```
<body>
<script type="text/javascript">
    var dt = new Date( "December 25, 1995 23:15:20" );
    document.write("getUTCFullYear() : " + dt.getUTCFullYear() );
</script>
</body>
</html>
```

Output

```
getUTCFullYear() : 1995
```

getUTCHours ()

Javascript date `getUTCHours()` method returns the hours in the specified date according to universal time. The value returned by `getUTCHours` is an integer between 0 and 23.

Syntax

Its syntax is as follows:

```
Date.getUTCHours ()
```

Return Value

Returns the hours in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCHours Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date();
    document.write("getUTCHours() : " + dt.getUTCHours() );
</script>
```

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

Output

```
getUTCHours() : 11
```

getUTCMilliseconds ()

Javascript date getUTCMilliseconds() method returns the milliseconds in the specified date according to universal time. The value returned by getUTCMilliseconds is an integer between 0 and 999.

Syntax

Its syntax is as follows:

```
Date.getUTCMilliseconds ()
```

Return Value

Returns the milliseconds in the specified date according to universal time.

Example

Try the following example.

```
<html>  
<head>  
<title>JavaScript getUTCMilliseconds Method</title>  
</head>  
<body>  
<script type="text/javascript">  
    var dt = new Date();  
    document.write("getUTCMilliseconds() : " + dt.getUTCMilliseconds() );  
</script>  
</body>  
</html>
```

Output

```
getUTCMilliseconds() : 206
```

getUTCMinutes ()

Javascript date getUTCMinutes() method returns the minutes in the specified date according to universal time. The value returned by getUTCMinutes is an integer between 0 and 59.

Syntax

Its syntax is as follows:

```
Date.getUTCMinutes ()
```

Return Value

Returns the minutes in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCMinutes Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date();
    document.write("getUTCMinutes() : " + dt.getUTCMinutes() );
</script>
</body>
</html>
```

Output

```
getUTCMinutes() : 18
```

getUTCMonth ()

Javascript date getUTCMonth() method returns the month in the specified date according to universal time. The value returned by getUTCMonth is an integer between 0 and 11 corresponding to the month. 0 for January, 1 for February, 2 for March, and so on.

Syntax

Its syntax is as follows:

```
Date.getUTCMonth ()
```

Return Value

Returns the month in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCMonth Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date();
    document.write("getUTCMonth() : " + dt.getUTCMonth() );
</script>
</body>
</html>
```

Output

```
getUTCMonth() : 2
```

getUTCSeconds ()

Javascript date getUTCSeconds() method returns the seconds in the specified date according to universal time. The value returned by getUTCSeconds is an integer between 0 and 59.

Syntax

Its syntax is as follows:

```
Date.getUTCSeconds ()
```

Return Value

Returns the month in the specified date according to universal time.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date();
    document.write("getUTCSeconds() : " + dt.getUTCSeconds() );
</script>
</body>
</html>
```

Output

```
getUTCSeconds() : 24
```

getYear ()

Javascript date getYear() method returns the year in the specified date according to universal time. The getYear is no longer used and has been replaced by the getFullYear method.

The value returned by getYear is the current year minus 1900. JavaScript 1.2 and earlier versions return either a 2-digit or 4-digit year. For example, if the year is 2026, the value returned is 2026. So before testing this function, you need to be sure of the javascript version you are using.

Syntax

Its syntax is as follows:

```
Date.getYear ()
```

Return Value

Returns the year in the specified date according to universal time.

Example

Try the following example.

```

<html>
<head>
<title>JavaScript getYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date();
    document.write("getYear() : " + dt.getYear() );
</script>
</body>
</html>

```

Output

```
getYear() : 115
```

setDate ()

Javascript date setDate() method sets the day of the month for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setDate( dayValue )
```

Parameter Detail

dayValue : An integer from 1 to 31, representing the day of the month.

Example

Try the following example.

```

<html>
<head>
<title>JavaScript setDate Method</title>
</head>
<body>
<script type="text/javascript">

```

```
var dt = new Date( "Aug 28, 2008 23:30:00" );  
dt.setDate( 24 );  
document.write( dt );  
</script>  
</body>  
</html>
```

Output

```
Sun Aug 24 2008 23:30:00 GMT+0530 (India Standard Time)
```

setFullYear ()

Javascript date setFullYear() method sets the full year for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setFullYear(yearValue[, monthValue[, dayValue]])
```

Parameter Detail

- **yearValue** : An integer specifying the numeric value of the year, for example, 2008.
- **monthValue** : An integer between 0 and 11 representing the months January through December.
- **dayValue** : An integer between 1 and 31 representing the day of the month. If you specify the dayValue parameter, you must also specify the monthValue.

If you do not specify the monthValue and day Value parameters, the values returned from the getMonth and getDate methods are used.

Example

Try the following example.

```
<html>
```

```
<head>
<title>JavaScript setFullYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setFullYear( 2000 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Mon Aug 28 2000 23:30:00 GMT+0530 (India Standard Time)
```

setHours ()

Javascript date setHours() method sets the hours for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setHours(hoursValue[, minutesValue[, secondsValue[, msValue]]])
```

Note: Parameters in the bracket are always optional.

Parameter Detail

- hoursValue : An integer between 0 and 23, representing the hour.
- minutesValue : An integer between 0 and 59, representing the minutes.
- secondsValue : An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.
- msValue : A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the minutesValue, secondsValue, and msValue parameters, the values returned from the getUTCMinutes, getUTCSeconds, and getMilliseconds methods are used.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setHours Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setHours( 02 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 02:30:00 GMT+0530 (India Standard Time)
```

setMilliseconds ()

JavaScript date setMilliseconds() method sets the milliseconds for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setMilliseconds(millisecondsValue)
```

Note: Parameters in the bracket are always optional .

Parameter Detail

millisecondsValue : A number between 0 and 999, representing the milliseconds.

If you specify a number outside the expected range, the date information in the Date object is updated accordingly. For example, if you specify 1010, the number of seconds is incremented by 1, and 10 is used for the milliseconds.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setMilliseconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setMilliseconds( 1010 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time)
```

setMinutes ()

Javascript date setMinutes() method sets the minutes for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setMinutes(minutesValue[, secondsValue[, msValue]])
```

Note: Parameters in the bracket are always optional.

Parameter Detail

- minutesValue : An integer between 0 and 59, representing the minutes.

- `secondsValue` : An integer between 0 and 59, representing the seconds. If you specify the `secondsValue` parameter, you must also specify the `minutesValue`.
- `msValue` : A number between 0 and 999, representing the milliseconds. If you specify the `msValue` parameter, you must also specify the `minutesValue` and `secondsValue`.

If you do not specify the `secondsValue` and `msValue` parameters, the values returned from `getSeconds` and `getMilliseconds` methods are used.

Try the following example.

```
<html>
<head>
<title>JavaScript setMinutes Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setMinutes( 45 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 23:45:00 GMT+0530 (India Standard Time)
```

setMonth()

Javascript date `setMonth()` method sets the month for a specified date according to local time.

Syntax

The following syntax for `setMonth()` Method.

```
Date.setMonth(monthValue[, dayValue])
```

Note: Parameters in the bracket are always optional .

Parameter Detail

- `monthValue` : An integer between 0 and 11 (representing the months January through December).
- `dayValue` : An integer from 1 to 31, representing the day of the month.
- `msValue` : A number between 0 and 999, representing the milliseconds. If you specify the `msValue` parameter, you must also specify the `minutesValue` and `secondsValue`.

If you do not specify the `dayValue` parameter, the value returned from the `getDate` method is used. If a parameter you specify is outside of the expected range, `setMonth` attempts to update the date information in the `Date` object accordingly. For example, if you use 1 5 for `monthValue`, the year will be incremented by 1 (year + 1), and 3 will be used for month.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setMonth Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setMonth( 2 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Fri Mar 28 2008 23:30:00 GMT+0530 (India Standard Time)
```

setSeconds ()

Javascript date `setSeconds()` method sets the seconds for a specified date according to local time.

Syntax

Its syntax is as follows:

```
Date.setSeconds(secondsValue[, msValue])
```

Note: Parameters in the bracket are always optional .

Parameter Detail

- secondsValue : An integer between 0 and 59.
- msValue : A number between 0 and 999, representing the milliseconds.

If you do not specify the msValue parameter, the value returned from the getMilliseconds method is used. If a parameter you specify is outside of the expected range, setSeconds attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes stored in the Date object will be incremented by 1, and 40 will be used for seconds.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setSeconds( 80 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 23:31:20 GMT+0530 (India Standard Time)
```

setTime ()

Javascript date setTime() method sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC.

Syntax

Its syntax is as follows:

```
Date.setTime(timeValue)
```

Note: Parameters in the bracket are always optional .

Parameter Detail

timeValue :An integer representing the number of milliseconds since 1 January 1970, 00:00:00 UTC.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setTime Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setTime( 5000000 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Jan 01 1970 06:53:20 GMT+0530 (India Standard Time)
```

setUTCDate ()

Javascript date setUTCDate() method sets the day of the month for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCDate(dayValue)
```

Note: Parameters in the bracket are always optional .

Parameter Detail

dayValue : An integer from 1 to 31, representing the day of the month.

If a parameter you specify is outside the expected range, setUTCDate attempts to update the date information in the Date object accordingly.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCDate Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCDate( 20 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Wed Aug 20 2008 23:30:00 GMT+0530 (India Standard Time)
```

setUTCFullYear ()

Javascript date setUTCFullYear() method sets the full year for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCFullYear(yearValue[, monthValue[, dayValue]])
```

Note: Parameters in the bracket are always optional .

Parameter Detail

- **yearValue** : An integer specifying the numeric value of the year, for example, 2008.
- **monthValue** : An integer between 0 and 11 representing the months January through December.
- **dayValue** : An integer between 1 and 31 representing the day of the month. If you specify the dayValue parameter, you must also specify the monthValue.

If you do not specify the monthValue and dayValue parameters, the values returned from the `getMonth` and `getDate` methods are used. If a parameter you specify is outside of the expected range, `setUTCFullYear` attempts to update the other parameters and the date information in the Date object accordingly. For example, if you specify 15 for monthValue, the year is incremented by 1 (year + 1), and 3 is used for the month.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCFullYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCFullYear( 2006 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Mon Aug 28 2006 23:30:00 GMT+0530 (India Standard Time)
```


setUTCHours ()

Javascript date setUTCHours() method sets the hour for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCHours(hoursValue[, minutesValue[, secondsValue[, msValue]]])
```

Note: Parameters in the bracket are always optional.

Parameter Detail

- hoursValue : An integer between 0 and 23, representing the hour.
- minutesValue : An integer between 0 and 59, representing the minutes.
- secondsValue : An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.
- msValue : A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the minutes Value, secondsValue, and msValue parameters, the values returned from the getUTCMinutes, getUTCSeconds, and getUTCMilliseconds methods are used.

If a parameter you specify is outside the expected range, setUTCHours attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes will be incremented by 1 (min + 1), and 40 will be used for seconds.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCHours Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 23:30:00" );
    dt.setUTCHours( 15 );
```

```
document.write( dt );  
</script>  
</body>  
</html>
```

Output

```
Thu Aug 28 2008 20:30:00 GMT+0530 (India Standard Time)
```

setUTCMilliseconds ()

Javascript date setUTCMilliseconds() method sets the milliseconds for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCMilliseconds(millisecondsValue)
```

Note: Parameters in the bracket are always optional .

Parameter Detail

millisecondsValue : A number between 0 and 999, representing the milliseconds.

If a parameter you specify is outside the expected range, setUTC Milliseconds attempts to update the date information in the Date object accordingly. For example, if you use 1100 for millisecondsValue, the seconds stored in the Date object will be incremented by 1, and 100 will be used for milliseconds.

Example

Try the following example.

```
<html>  
<head>  
<title>JavaScript setUTCMilliseconds Method</title>  
</head>  
<body>  
<script type="text/javascript">  
    var dt = new Date( "Aug 28, 2008 23:30:00" );  
    dt.setUTCMilliseconds( 1100 );
```

```

    document.write( dt );
</script>
</body>
</html>

```

Output

```
Thu Aug 28 2008 23:30:01 GMT+0530 (India Standard Time)
```

setUTCMinutes ()

Javascript date setUTCMinutes() method sets the minutes for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCMinutes(minutesValue[, secondsValue[, msValue]])
```

Note: Parameters in the bracket are always optional .

Parameter Detail

- **minutesValue** : An integer between 0 and 59, representing the minutes.
- **secondsValue** : An integer between 0 and 59, representing the seconds. If you specify the secondsValue parameter, you must also specify the minutesValue.
- **msValue** : A number between 0 and 999, representing the milliseconds. If you specify the msValue parameter, you must also specify the minutesValue and secondsValue.

If you do not specify the secondsValue and msValue parameters, the values returned from getUTCSeconds and getUTCMilliseconds methods are used.

If a parameter you specify is outside of the expected range, setUTCMinutes attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes (minutesValue) will be incremented by 1 (minutesValue + 1), and 40 will be used for seconds.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCMinutes Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 13:30:00" );
    dt.setUTCMinutes( 65 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 14:35:00 GMT+0530 (India Standard Time)
```

setUTC Month ()

Javascript date setUTC Month () method sets the month for a specified date according to universal time.

Syntax

The following syntax for setUTCMonth () Method.

```
Date.setUTCMonth ( monthvalue )
```

Note: Parameters in the bracket are always optional .

Parameter Detail

month Value : An integer between 0 and 11, representing the month .

Example

Try the following example.

```
<html>
<head>
<title>JavaScript getUTCSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 13:30:00" );
    dt.setUTCMonth( 2 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Fri Mar 28 2008 13:30:00 GMT+0530 (India Standard Time)
```

setUTCSeconds ()

Javascript date setUTCSeconds() method sets the seconds for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setUTCSeconds(secondsValue[, msValue])
```

Note: Parameters in the bracket are always optional .

Parameter Detail

- secondsValue : An integer between 0 and 59, representing the seconds.
- msValue : A number between 0 and 999, representing the milliseconds.

If you do not specify the msValue parameter, the value returned from the getUTCMilliseconds methods is used.

If a parameter you specify is outside the expected range, setUTCSeconds attempts to update the date information in the Date object accordingly. For example, if you use 100 for secondsValue, the minutes stored in the Date object will be incremented by 1, and 40 will be used for seconds.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript setUTCSeconds Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 13:30:00" );
    dt.setUTCSeconds( 65 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Thu Aug 28 2008 13:31:05 GMT+0530 (India Standard Time)
```

setYear ()

Javascript date setYear() method sets the year for a specified date according to universal time.

Syntax

Its syntax is as follows:

```
Date.setYear(yearValue)
```

Note: Parameters in the bracket are always optional .

Parameter Detail

yearValue : An integer value.

Example

Try the following example.

```
<html>
```

```
<head>
<title>JavaScript setYear Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date( "Aug 28, 2008 13:30:00" );
    dt.setYear( 2000 );
    document.write( dt );
</script>
</body>
</html>
```

Output

```
Mon Aug 28 2000 13:30:00 GMT+0530 (India Standard Time)
```

toDateDateString ()

Javascript date toDateDateString() method returns the date portion of a Date object in human readable form.

Syntax

Its syntax is as follows:

```
Date.toDateDateString()
```

Return Value

Returns the date portion of a Date object in human readable form.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toDateDateString Method</title>
</head>
<body>
<script type="text/javascript">
```

```
var dt = new Date(1993, 6, 28, 14, 39, 7);  
document.write( "Formatted Date : " + dt.toString() );  
</script>  
</body>  
</html>
```

Output

```
Formatted Date : Wed Jul 28 1993
```

toGMTString ()

Javascript date toGMTString() method converts a date to a string, using Internet GMT conventions.

This method is no longer used and has been replaced by the toUTCString method.

Syntax

Its syntax is as follows:

```
Date.toGMTString()
```

Return Value

Returns a date to a string, using Internet GMT conventions.

Example

Try the following example.

```
<html>  
<head>  
<title>JavaScript toGMTString Method</title>  
</head>  
<body>  
<script type="text/javascript">  
    var dt = new Date(1993, 6, 28, 14, 39, 7);  
    document.write( "Formatted Date : " + dt.toGMTString() );  
</script>  
</body>
```



```
</html>
```

Output

```
Formatted Date : Wed, 28 Jul 1993 09:09:07 GMT
```

toLocaleDateString ()

Javascript date toLocaleDateString() method converts a date to a string, returning the "date" portion using the operating system's locale's conventions.

Syntax

Its syntax is as follows:

```
Date.toGMTString()
```

Return Value

Returns a date to a string, using Internet GMT conventions.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toGMTString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toGMTString() );
</script>
</body>
</html>
```

Output

```
Formatted Date : Wed, 28 Jul 1993 09:09:07 GMT
```

toLocaleDateString ()

Javascript date `toLocaleDateString()` method converts a date to a string, returning the "date" portion using the operating system's locale's conventions.

Syntax

Its syntax is as follows:

```
Date.toLocaleString()
```

Return Value

Returns the "date" portion using the operating system's locale's conventions.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleDateString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toLocaleDateString() );
</script>
</body>
</html>
```

Output

```
Formatted Date : 7/28/1993
```

toLocaleFormat ()

Javascript date `toLocaleFormat()` method converts a date to a string using the specified formatting.

Note: This method may not be compatible with all the browsers.

Syntax

Its syntax is as follows:

```
Date.toLocaleFormat()
```

Parameter Details

formatString : A format string in the same format expected by the strftime() function in C.

Return Value

Returns the formatted date.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleFormat Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toLocaleFormat( "%A, %B %e, %Y" ));
</script>
</body>
</html>
```

Output

```
Formatted Date : Wed Jul 28 1993 14:39:07 GMT+0530 (India Standard Time)
```

toLocaleString ()

Javascript date toLocaleString() method converts a date to a string, using the operating system's local conventions.

The toLocaleString method relies on the underlying operating system in formatting dates. It converts the date to a string using the formatting convention of the operating system where the script is running. For example, in the United States, the month appears before the date (04/15/98), whereas in Germany the date appears before the month (15.04.98).

Syntax

Its syntax is as follows:

```
Date.toLocaleString ()
```

Return Value

Returns the formatted date in a string format.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toLocaleString() );
</script>
</body>
</html>
```

Output

```
Formatted Date : 7/28/1993, 2:39:07 PM
```

toLocaleTimeString ()

Javascript date toLocaleTimeString() method converts a date to a string, returning the "date" portion using the current locale's conventions.

The toLocaleTimeString method relies on the underlying operating system in formatting dates. It converts the date to a string using the formatting convention of the operating system where the script is running. For example, in the United States, the month appears before the date (04/15/98), whereas in Germany, the date appears before the month (15.04.98).

Syntax

Its syntax is as follows:

```
Date.toLocaleTimeString ()
```

Return Value

Returns the formatted date in a string format.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toLocaleTimeString Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toLocaleTimeString() );
</script>
</body>
</html>
```

Output

```
Formatted Date : 2:39:07 PM
```

toSource ()

This method returns a string representing the source code of the object.

Note: This method may not be compatible with all the browsers.

Syntax

The following syntax for toSource () Method.

```
Date.toSource ()
```

Return Value

- For the built-in Date object, toSource returns a string (new Date(...)) indicating that the source code is not available

- For instances of Date, toSource returns a string representing the source code.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toSource Method</title>
</head>
<body>
<script type="text/javascript">
    var dt = new Date(1993, 6, 28, 14, 39, 7);
    document.write( "Formatted Date : " + dt.toSource() );
</script>
</body>
</html>
```

Output

```
Formatted Date : (new Date(743850547000))
```

toString ()

This method returns a string representing the specified Date object.

Syntax

The following syntax for toString () Method.

```
Date.toString ()
```

Return Value

Returns a string representing the specified Date object .

Example

Try the following example.

```
<html>
```

```
<head>
<title>JavaScript toString Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    stringobj = dateobject.toString();
    document.write( "String Object : " + stringobj );
</script>
</body>
</html>
```

Output

```
String Object : Wed Jul 28 1993 14:39:07 GMT+0530 (India Standard Time)
```

toString ()

This method returns the time portion of a Date object in human readable form.

Syntax

Its syntax is as follows:

```
Date.toString ()
```

Return Value

Returns the time portion of a Date object in human readable form.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toString Method</title>
```

```
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    document.write( dateobject.toTimeString() );
</script>
</body>
</html>
```

Output

```
14:39:07 GMT+0530 (India Standard Time)
```

toUTCString ()

This method converts a date to a string, using the universal time convention.

Syntax

Its syntax is as follows:

```
Date.toTimeString ()
```

Return Value

Returns converted date to a string, using the universal time convention.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript toUTCString Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    document.write( dateobject.toUTCString() );
</script>
</body>
```



```
</html>
```

Output

```
Wed, 28 Jul 1993 09:09:07 GMT
```

valueOf ()

This method returns the primitive value of a Date object as a number data type, the number of milliseconds since midnight 01 January, 1970 UTC.

Syntax

Its syntax is as follows:

```
Date.valueOf ()
```

Return Value

Returns the primitive value of a Date object.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript valueOf Method</title>
</head>
<body>
<script type="text/javascript">
    var dateobject = new Date(1993, 6, 28, 14, 39, 7);
    document.write( dateobject.valueOf() );
</script>
</body>
</html>
```

Output

743850547000

Date Static Methods

In addition to the many instance methods listed previously, the Date object also defines two static methods. These methods are invoked through the Date () constructor itself.

Method	Description
Date.parse()	Parses a string representation of a date and time and returns the internal millisecond representation of that date.
Date.UTC()	Returns the millisecond representation of the specified UTC date and time.

In the following sections, we will have a few examples to demonstrate the usages of Date Static methods.

Date.parse ()

Javascript date parse() method takes a date string and returns the number of milliseconds since midnight of January 1, 1970.

Syntax

Its syntax is as follows:

Date.parse(datestring)

Note: Parameters in the bracket are always optional .

Parameter Details

datestring : A string representing a date .

Return Value

Number of milliseconds since midnight of January 1, 1970.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript parse Method</title>
</head>
<body>
<script type="text/javascript">
    var msecs = Date.parse( "Aug 28, 2008 23:30:00" );
    document.write( "Number of milliseconds from 1970: " + msecs );
</script>
</body>
</html>
```

Output

```
Number of milliseconds from 1970: 1219946400000
```

Date.UTC()

This method takes a date and returns the number of milliseconds since midnight of January 1, 1970 according to universal time.

Syntax

Its syntax is as follows:

```
Date.year,month,day,[hours,[minutes,[seconds,[ms]]]])
```

Note: Parameters in the bracket are always optional .

Parameter Details

- year : A four digit number representing the year .
- month : An integer between 0 and 11 representing the month .
- day : An integer between 1 and 31 representing the date .
- hours : An integer between 0 and 23 representing the hour .

- minutes : An integer between 0 and 59 representing the minutes .
- seconds : An integer between 0 and 59 representing the seconds.
- ms : An integer between 0 and 999 representing the milliseconds .

Return Value

Number of milliseconds since midnight of January 1, 1970.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript UTC Method</title>
</head>
<body>
<script type="text/javascript">
    var msec = Date.UTC(2008,9,6);
    document.write( "Number of milliseconds from 1970: " + msec );
</script>
</body>
</html>
```

Output

```
Number of milliseconds from 1970: 1223251200000
```

26. MATH

The `math` object provides you properties and methods for mathematical constants and functions. Unlike other global objects, `Math` is not a constructor. All the properties and methods of `Math` are static and can be called by using `Math` as an object without creating it.

Thus, you refer to the constant `pi` as `Math.PI` and you call the *sine* function as `Math.sin(x)`, where `x` is the method's argument.

Syntax

The syntax to call the properties and methods of `Math` are as follows:

```
var pi_val = Math.PI;  
var sine_val = Math.sin(30);
```

Math Properties

Here is a list of all the properties of `Math` and their description.

Property	Description
E	Euler's constant and the base of natural logarithms, approximately 2.718.
LN2	Natural logarithm of 2, approximately 0.693.
LN10	Natural logarithm of 10, approximately 2.302.
LOG2E	Base 2 logarithm of E, approximately 1.442.
LOG10E	Base 10 logarithm of E, approximately 0.434.
PI	Ratio of the circumference of a circle to its diameter, approximately 3.14159.
SQRT1_2	Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707.
SQRT2	Square root of 2, approximately 1.414.

In the following sections, we will have a few examples to demonstrate the usage of Math properties.

Math-E

This is an Euler's constant and the base of natural logarithms, approximately 2.718.

Syntax

Its syntax is as follows:

```
Math.E
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math E Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.E
    document.write("Property Value is :" + property_value);
</script>
</body>
</html>
```

Output

```
Property Value is :2.718281828459045
```

Math.LN2

It returns the natural logarithm of 2 which is approximately 0.693.

Syntax

Its syntax is as follows:

```
Math.LN2
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math LN2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LN2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

```
Property Value is : 0.6931471805599453
```

Math.LN10

It returns the natural logarithm of 10 which is approximately 2.302.

Syntax

Its syntax is as follows:

```
Math.LN10
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math LN10 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LN10
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

```
Property Value is : 2.302585092994046
```

Math-LOG2E

It returns the base 2 logarithm of E which is approximately 1.442.

Syntax

Its syntax is as follows:

```
Math.LOG2E
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math LOG2E Property</title>
</head>
<body>
<script type="text/javascript">
```



```
var property_value = Math.LOG2E
document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

Property Value is : 1.4426950408889634

Math-LOG10E

It returns the base 10 logarithm of E which is approximately 0.434.

Syntax

Its syntax is as follows:

```
Math.LOG10E
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math LOG10E Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.LOG10E
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

Property Value is : 0.4342944819032518

Math-PI

It returns the ratio of the circumference of a circle to its diameter which is approximately 3.14159.

Syntax

Its syntax is as follows:

Math.PI

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math PI Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.PI
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

Property Value is : 3.141592653589793

Math-SQRT1_2

It returns the square root of 1/2; equivalently, 1 over the square root of 2 which is approximately 0.707.

Syntax

Its syntax is as follows:

```
Math.SQRT1_2
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math SQRT1_2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.SQRT1_2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>
```

Output

```
Property Value is : 0.7071067811865476
```

Math-SQRT2

It returns the square root of 2 which is approximately 1.414.

Syntax

Its syntax is as follows:

```
Math.SQRT2
```

Example

Try the following example program.

```
<html>
```

```

<head>
<title>JavaScript Math SQRT2 Property</title>
</head>
<body>
<script type="text/javascript">
    var property_value = Math.SQRT2
    document.write("Property Value is : " + property_value);
</script>
</body>
</html>

```

Output

```
Property Value is : 1.4142135623730951
```

Math Methods

Here is a list of the methods associated with Math object and their description.

Method	Description
abs()	Returns the absolute value of a number.
acos()	Returns the arccosine (in radians) of a number.
asin()	Returns the arcsine (in radians) of a number.
atan()	Returns the arctangent (in radians) of a number.
atan2()	Returns the arctangent of the quotient of its arguments.
ceil()	Returns the smallest integer greater than or equal to a number.
cos()	Returns the cosine of a number.
exp()	Returns E^N , where N is the argument, and E is Euler's constant, the base of the natural logarithm.
floor()	Returns the largest integer less than or equal to a

	number.
log()	Returns the natural logarithm (base E) of a number.
max()	Returns the largest of zero or more numbers.
min()	Returns the smallest of zero or more numbers.
pow()	Returns base to the exponent power, that is, base exponent.
random()	Returns a pseudo -random number between 0 and 1.
round()	Returns the value of a number rounded to the nearest integer.
sin()	Returns the sine of a number.
sqrt()	Returns the square root of a number.
tan()	Returns the tangent of a number.
toSource()	Returns the string "Math".

In the following sections, we will have a few examples to demonstrate the usage of the methods associated with Math.

abs()

This method returns the absolute value of a number.

Syntax

Its syntax is as follows:

```
Math.abs( x ) ;
```

Parameter Details

x: A number.

Return Value

Returns the absolute value of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math abs() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.abs(-1);
    document.write("First Test Value : " + value );

    var value = Math.abs(null);
    document.write("<br />Second Test Value : " + value );

    var value = Math.abs(20);
    document.write("<br />Third Test Value : " + value );

    var value = Math.abs("string");
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 1
Second Test Value : 0
Third Test Value : 20
Fourth Test Value : NaN
```

acos ()

This method returns the arccosine in radians of a number. The acos method returns a numeric value between 0 and pi radians for x between -1 and 1. If the value of number is outside this range, it returns NaN.

Syntax

Its syntax is as follows:

```
Math.cos( x ) ;
```

Parameter Details

x: A number.

Return Value

Returns the arccosine in radians of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math acos() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.acos(-1);
    document.write("First Test Value : " + value );

    var value = Math.acos(null);
    document.write("<br />Second Test Value : " + value );

    var value = Math.acos(30);
    document.write("<br />Third Test Value : " + value );

    var value = Math.acos("string");
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 3.141592653589793
Second Test Value : 1.5707963267948966
Third Test Value : NaN
Fourth Test Value : NaN
```

asin ()

This method returns the arcsine in radians of a number. The asin method returns a numeric value between $-\pi/2$ and $\pi/2$ radians for x between -1 and 1. If the value of number is outside this range, it returns NaN.

Syntax

Its syntax is as follows:

```
Math.asin( x ) ;
```

Parameter Details

x: A number.

Return Value

Returns the arcsine in radians of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math asin() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.asin(-1);
    document.write("First Test Value : " + value );

    var value = Math.asin(null);
    document.write("<br />Second Test Value : " + value );
```



```
var value = Math.asin(30);  
document.write("<br />Third Test Value : " + value );  
  
var value = Math.asin("string");  
document.write("<br />Fourth Test Value : " + value );  
</script>  
</body>  
</html>
```

Output

```
First Test Value : -1.5707963267948966  
Second Test Value : 0  
Third Test Value : NaN  
Fourth Test Value : NaN
```

atan ()

This method returns the arctangent in radians of a number. The atan method returns a numeric value between $-\pi/2$ and $\pi/2$ radians.

Syntax

Its syntax is as follows:

```
Math.atan( x ) ;
```

Parameter Details

x: A number.

Return Value

Returns the arctangent in radians of a number.

Example

Try the following example program.

```
<html>  
<head>
```

```
<title>JavaScript Math atan() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.atan(-1);
    document.write("First Test Value : " + value );

    var value = Math.atan(.5);
    document.write("<br />Second Test Value : " + value );

    var value = Math.atan(30);
    document.write("<br />Third Test Value : " + value );

    var value = Math.atan("string");
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : -0.7853981633974483
Second Test Value : 0.4636476090008061
Third Test Value : 1.5374753309166493
Fourth Test Value : NaN
```

atan2()

This method returns the arctangent of the quotient of its arguments. The atan2 method returns a numeric value between $-\pi$ and π representing the angle theta of an (x, y) point.

Syntax

Its syntax is as follows:

```
Math.atan2 ( x, y ) ;
```

Parameter Details

X and y : numbers.

Return Value

Returns the arctangent in radians of a number.

```
Math.atan2 ( ±0, -0 ) returns ±PI.
Math.atan2 ( ±0, +0 ) returns ±0.
Math.atan2 ( ±0, -x ) returns ±PI for x < 0.
Math.atan2 ( ±0, x ) returns ±0 for x > 0.
Math.atan2 ( y, ±0 ) returns -PI/2 for y > 0.
Math.atan2 ( ±y, -Infinity ) returns ±PI for finite y > 0.
Math.atan2 ( ±y, +Infinity ) returns ±0 for finite y > 0.
Math.atan2 ( ±Infinity, +x ) returns ±PI/2 for finite x.
Math.atan2 ( ±Infinity, -Infinity ) returns ±3*PI/4.
Math.atan2 ( ±Infinity, +Infinity ) returns ±PI/4.
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math atan2() Method</title>
</head>
<body>
<script type="text/javascript">
    var value = Math.atan2(90,15);
    document.write("First Test Value : " + value );

    var value = Math.atan2(15,90);
    document.write("<br />Second Test Value : " + value );

    var value = Math.atan2(0, -0);
    document.write("<br />Third Test Value : " + value );
```

```
var value = Math.atan2(+Infinity, -Infinity);  
document.write("<br />Fourth Test Value : " + value );  
</script>  
</body>  
</html>
```

Output

```
First Test Value : 1.4056476493802699  
Second Test Value : 0.16514867741462683  
Third Test Value : 3.141592653589793  
Fourth Test Value : 2.356194490192345
```

ceil ()

This method returns the smallest integer greater than or equal to a number.

Syntax

Its syntax is as follows:

```
Math.ceil ( x ) ;
```

Parameter Details

x: a number.

Return Value

Returns the smallest integer greater than or equal to a number.

Example

Try the following example program.

```
<html>  
<head>  
<title>JavaScript Math ceil() Method</title>  
</head>  
<body>  
<script type="text/javascript">
```

```
var value = Math.ceil(45.95);  
document.write("First Test Value : " + value );  
  
var value = Math.ceil(45.20);  
document.write("<br />Second Test Value : " + value );  
  
var value = Math.ceil(-45.95);  
document.write("<br />Third Test Value : " + value );  
  
var value = Math.ceil(-45.20);  
document.write("<br />Fourth Test Value : " + value );  
</script>  
</body>  
</html>
```

Output

```
First Test Value : 46  
Second Test Value : 46  
Third Test Value : -45  
Fourth Test Value : -45
```

cos()

This method returns the cosine of a number. The cos method returns a numeric value between -1 and 1, which represents the cosine of the angle.

Syntax

Its syntax is as follows:

```
Math.cos ( x ) ;
```

Parameter Details

x: a number.

Return Value

Returns the cosine of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math cos() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.cos(90);
    document.write("First Test Value : " + value );

    var value = Math.cos(30);
    document.write("<br />Second Test Value : " + value );

    var value = Math.cos(-1);
    document.write("<br />Third Test Value : " + value );

    var value = Math.cos(2*Math.PI);
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : -0.4480736161291702
Second Test Value : 0.15425144988758405
Third Test Value : 0.5403023058681398
Fourth Test Value : 1
```

exp()

This method returns E^x , where x is the argument, and E is the Euler's constant, the base of the natural logarithms.

Syntax

Its syntax is as follows:

```
Math.exp ( x ) ;
```

Parameter Details

x: a number .

Return Value

Returns the exponential value of the variable x.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math exp() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.exp(1);
    document.write("First Test Value : " + value );

    var value = Math.exp(30);
    document.write("<br />Second Test Value : " + value );

    var value = Math.exp(-1);
    document.write("<br />Third Test Value : " + value );

    var value = Math.exp(.5);
    document.write("<br />Fourth Test Value : " + value );

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

Output

```
First Test Value : 2.718281828459045
Second Test Value : 10686474581524.482
Third Test Value : 0.3678794411714424
Fourth Test Value : 1.6487212707001282
```

floor ()

This method returns the largest integer less than or equal to a number.

Syntax

Its syntax is as follows:

```
Math.floor ( x ) ;
```

Parameter Details

x: a number.

Return Value

Returns the largest integer less than or equal to a number x.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math floor() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.floor(10.3);
    document.write("First Test Value : " + value );

    var value = Math.floor(30.9);
    document.write("<br />Second Test Value : " + value );

    var value = Math.floor(-2.9);
```



```
document.write("<br />Third Test Value : " + value );

var value = Math.floor(-2.2);
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 10
Second Test Value : 30
Third Test Value : -3
Fourth Test Value : -3
```

log()

This method returns the natural logarithm (base E) of a number. If the value of number is negative, the return value is always NaN.

Syntax

Its syntax is as follows:

```
Math.log ( x ) ;
```

Parameter Details

x: a number.

Return Value

Returns the natural logarithm (base E) of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math log() Method</title>
</head>
<body>
```

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

    var value = Math.log(10);
    document.write("First Test Value : " + value );

    var value = Math.log(0);
    document.write("<br />Second Test Value : " + value );

    var value = Math.log(-1);
    document.write("<br />Third Test Value : " + value );

    var value = Math.log(100);
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 2.302585092994046
Second Test Value : -Infinity
Third Test Value : NaN
Fourth Test Value : 4.605170185988092
```

max()

This method returns the largest of zero or more numbers. If no arguments are given, the results is `Infinity`.

Syntax

Its syntax is as follows:

```
Math.max(value1, value2, ... valueN ) ;
```

Parameter Details

value1, value2, ... valueN : Numbers.

Return Value

Returns the largest of zero or more numbers.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math max() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.max(10, 20, -1, 100);
    document.write("First Test Value : " + value );

    var value = Math.max(-1, -3, -40);
    document.write("<br />Second Test Value : " + value );

    var value = Math.max(0, -1);
    document.write("<br />Third Test Value : " + value );

    var value = Math.max(100);
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 100
Second Test Value : -1
Third Test Value : 0
Fourth Test Value : 100
```

min()

This method returns the smallest of zero or more numbers. If no arguments are given, the results is `+Infinity`.

Syntax

Its syntax is as follows:

```
Math.min (value1, value2, ... valueN ) ;
```

Parameter Details

value1, value2, ... valueN : Numbers.

Return Value

Returns the smallest of zero or more numbers.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math min() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.min(10, 20, -1, 100);
    document.write("First Test Value : " + value );

    var value = Math.min(-1, -3, -40);
    document.write("<br />Second Test Value : " + value );

    var value = Math.min(0, -1);
    document.write("<br />Third Test Value : " + value );

    var value = Math.min(100);
    document.write("<br />Fourth Test Value : " + value );
```

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

Output

```
First Test Value : -1  
Second Test Value : -40  
Third Test Value : -1  
Fourth Test Value : 100
```

pow ()

This method returns the base to the exponent power, that is, $\text{base}^{\text{exponent}}$.

Syntax

Its syntax is as follows:

```
Math.pow(base, exponent );
```

Parameter Details

- **base** : The base number.
- **exponents** : The exponent to which to raise the base.

Return Value

Returns the base to the exponent power, that is, $\text{base}^{\text{exponent}}$.

Example

Try the following example program.

```
<html>  
<head>  
<title>JavaScript Math pow() Method</title>  
</head>  
<body>  
<script type="text/javascript">
```

```
var value = Math.pow(7, 2);  
document.write("First Test Value : " + value );  
  
var value = Math.pow(8, 8);  
document.write("<br />Second Test Value : " + value );  
  
var value = Math.pow(-1, 2);  
document.write("<br />Third Test Value : " + value );  
  
var value = Math.pow(0, 10);  
document.write("<br />Fourth Test Value : " + value );  
</script>  
</body>  
</html>
```

Output

```
First Test Value : 49  
Second Test Value : 16777216  
Third Test Value : 1  
Fourth Test Value : 0
```

random ()

This method returns a random number between 0 (inclusive) and 1 (exclusive) .

Syntax

Its syntax is as follows:

```
Math.random ( );
```

Return Value

Returns a random number between 0 (inclusive) and 1 (exclusive).

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math random() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.random( );
    document.write("First Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Second Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Third Test Value : " + value );

    var value = Math.random( );
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 0.4093269258737564
Second Test Value : 0.023646741174161434
Third Test Value : 0.2672571325674653
Fourth Test Value : 0.38755513448268175
```

round ()

This method returns the value of a number rounded to the nearest integer.

Syntax

Its syntax is as follows:

```
Math.round ( );
```

Return Value

Returns the value of a number rounded to the nearest integer.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math round() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.round( 0.5 );
    document.write("First Test Value : " + value );

    var value = Math.round( 20.7 );
    document.write("<br />Second Test Value : " + value );

    var value = Math.round( 20.3 );
    document.write("<br />Third Test Value : " + value );

    var value = Math.round( -20.3 );
    document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 1
Second Test Value : 21
Third Test Value : 20
Fourth Test Value : -20
```


sin()

This method returns the sine of a number. The `sin` method returns a numeric value between -1 and 1, which represents the sine of the argument.

Syntax

Its syntax is as follows:

```
Math.sin ( x );
```

Parameter Details

x: A number.

Return Value

Returns the sine of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math sin() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.sin( 0.5 );
    document.write("First Test Value : " + value );

    var value = Math.sin( 90 );
    document.write("<br />Second Test Value : " + value );

    var value = Math.sin( 1 );
    document.write("<br />Third Test Value : " + value );

    var value = Math.sin( Math.PI/2 );
    document.write("<br />Fourth Test Value : " + value );
```

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

Output

```
First Test Value : 0.479425538604203  
Second Test Value : 0.8939966636005579  
Third Test Value : 0.8414709848078965  
Fourth Test Value : 1
```

sqrt()

This method returns the square root of a number. If the value of a number is negative, sqrt returns NaN.

Syntax

Its syntax is as follows:

```
Math.sqrt ( x );
```

Parameter Details

x: A number.

Return Value

Returns the square root of a given number.

Example

Try the following example program.

```
<html>  
<head>  
<title>JavaScript Math sqrt() Method</title>  
</head>  
<body>  
<script type="text/javascript">
```

```
var value = Math.sqrt( 0.5 );  
document.write("First Test Value : " + value );  
  
var value = Math.sqrt( 81 );  
document.write("<br />Second Test Value : " + value );  
  
var value = Math.sqrt( 13 );  
document.write("<br />Third Test Value : " + value );  
  
var value = Math.sqrt( -4 );  
document.write("<br />Fourth Test Value : " + value );  
</script>  
</body>  
</html>
```

Output

```
First Test Value : 0.7071067811865476  
Second Test Value : 9  
Third Test Value : 3.605551275463989  
Fourth Test Value : NaN
```

tan()

This method returns the tangent of a number. The tan method returns a numeric value that represents the tangent of the angle.

Syntax

Its syntax is as follows:

```
Math.tan ( x );
```

Parameter Details

x: A number representing an angle in radians .

Return Value

Returns the tangent of a number.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math tan() Method</title>
</head>
<body>
<script type="text/javascript">

var value = Math.tan( -30 );
document.write("First Test Value : " + value );

var value = Math.tan( 90 );
document.write("<br />Second Test Value : " + value );

var value = Math.tan( 45 );
document.write("<br />Third Test Value : " + value );

var value = Math.tan( Math.PI/180 );
document.write("<br />Fourth Test Value : " + value );
</script>
</body>
</html>
```

Output

```
First Test Value : 1
Second Test Value : 21
Third Test Value : 20
Fourth Test Value : -20
```

toSource ()

This method returns the string "Math". But this method does not work with IE.

Syntax

Its syntax is as follows:

```
Math.toSource ( );
```

Return Value

```
F Y h i f b g ` h \ Y ` g h f ] b [ ` í A U h \ í "
```

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript Math toSource() Method</title>
</head>
<body>
<script type="text/javascript">

    var value = Math.toSource( );
    document.write("Value : " + value );

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

Output

```
Value : Math
```

27. REGEXP

A regular expression is an object that describes a pattern of characters.

The JavaScript `RegExp` class represents regular expressions, and both `String` and `RegExp` define methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.

Syntax

A regular expression could be defined with the `RegExp()` constructor, as follows:

```
var pattern = new RegExp(pattern, attributes);
```

or simply

```
var pattern = /pattern/attributes;
```

Here is the description of the parameters:

- **pattern:** A string that specifies the pattern of the regular expression or another regular expression.
- **attributes:** An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

Brackets

Brackets (`[]`) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Expression	Description
<code>[...]</code>	Any one character between the brackets.
<code>[^...]</code>	Any one character not between the brackets.
<code>[0-9]</code>	It matches any decimal digit from 0 through 9.

[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character has a specific connotation. The +, *, ?, and \$ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing at least one p.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing one or more p's.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2, }	It matches any string containing a sequence of at least two p's.
p\$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Examples

Following examples explain more about matching characters.

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^. {2}\$	It matches any string containing exactly two characters.
(.)	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

Literal Characters

Character	Description
Alphanumeric	Itself
\0	The NUL character (\u0000)
\t	Tab (\u0009)
\n	Newline (\u000A)
\v	Vertical tab (\u000B)
\f	Form feed (\u000C)
\r	Carriage return (\u000D)
\xnn	The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n
\uxxxx	The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t

<code>\cX</code>	The control character <code>^X</code> ; for example, <code>\cJ</code> is equivalent to the newline character <code>\n</code>
------------------	--

Metacharacters

A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for a large sum of money using the `'\d'` metacharacter: `/([\d]+)000/`. Here `\d` will search for any string of numerical character.

The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

Character	Description
<code>.</code>	a single character
<code>\s</code>	a whitespace character (space, tab, newline)
<code>\S</code>	non-whitespace character
<code>\d</code>	a digit (0-9)
<code>\D</code>	a non-digit
<code>\w</code>	a word character (a-z, A-Z, 0-9, <code>_</code>)
<code>\W</code>	a non-word character
<code>[\b]</code>	a literal backspace (special case).
<code>[aeiou]</code>	matches a single character in the given set
<code>[^aeiou]</code>	matches a single character outside the given set
<code>(foo bar baz)</code>	matches any of the alternatives specified

Modifiers

Several modifiers are available that can simplify the way you work with regexps, like case sensitivity, searching in multiple lines, etc.

Modifier	Description
<code>i</code>	Performs case-insensitive matching.

m	Specifies that if the string has newline or carriage return characters, the ^ and \$ operators will now match against a newline boundary, instead of a string boundary
g	Performs a global match that is, find all matches rather than stopping after the first match.

RegExp Properties

Here is a list of the properties associated with RegExp and their description.

Property	Description
constructor	Specifies the function that creates an object's prototype.
global	Specifies if the "g" modifier is set.
ignoreCase	Specifies if the "i" modifier is set.
lastIndex	The index at which to start the next match.
multiline	Specifies if the "m" modifier is set.
source	The text of the pattern.

In the following sections, we will have a few examples to demonstrate the usage of RegExp properties.

constructor

It returns a reference to the array function that created the instance's prototype.

Syntax

Its syntax is as follows:

```
RegExp.constructor
```

Return Value

Returns the function that created this object's instance.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp constructor Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );
    document.write("re.constructor is:" + re.constructor);
</script>
</body>
</html>
```

Output

```
re.constructor is:function RegExp() { [native code]}
```

global

`global` is a read-only boolean property of `RegExp` objects. It specifies whether a particular regular expression performs global matching , i.e., whether it was created with the "g" attribute.

Syntax

Its syntax is as follows:

```
RegExpObject.global
```

Return Value

Returns "TRUE" if the "g" modifier is set, "FALSE" otherwise.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp global Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );

    if ( re.global ){
        document.write("Test1 - Global property is set");
    }else{
        document.write("Test1 - Global property is not set");
    }
    re = new RegExp( "string", "g" );

    if ( re.global ){
        document.write("<br />Test2 - Global property is set");
    }else{
        document.write("<br />Test2 - Global property is not set");
    }
</script>
</body>
</html>
```

Output

```
Test1 - Global property is not set
Test2 - Global property is set
```

ignoreCase

ignoreCase is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs case -insensitive matching , i.e., whether it was created with the "i" attribute.

Syntax

Its syntax is as follows:

```
RegExpObject.ignoreCase
```

Return Value

Returns "TRUE" if the "i" modifier is set, "FALSE" otherwise.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp ignoreCase Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );

    if ( re.ignoreCase ){
        document.write("Test1 - ignoreCase property is set");
    }else{
        document.write("Test1 - ignoreCase property is not set");
    }
    re = new RegExp( "string", "i" );

    if ( re.ignoreCase ){
        document.write("<br />Test2 - ignoreCase property is set");
    }else{
        document.write("<br />Test2 - ignoreCase property is not set");
    }
</script>
</body>
</html>
```

Output

```
Test1 - ignoreCase property is not set  
Test2 - ignoreCase property is set
```

lastIndex

`lastIndex` is a read/write property of `RegExp` objects. For regular expressions with the "g" attribute set, it contains an integer that specifies the character position immediately following the last match found by the `RegExp.exec()` and `RegExp.test()` methods. These methods use this property as the starting point for the next search they conduct.

This property allows you to call those methods repeatedly, to loop through all matches in a string and works only if the "g" modifier is set.

This property is read/write, so you can set it at any time to specify where in the target string, the next search should begin. `exec()` and `test()` automatically reset the `lastIndex` to 0 when they fail to find a match (or another match).

Syntax

Its syntax is as follows:

```
RegExpObject.lastIndex
```

Return Value

Returns an integer that specifies the character position immediately following the last match.

Example

Try the following example program.

```
<html>  
<head>  
<title>JavaScript RegExp lastIndex Property</title>  
</head>  
<body>  
<script type="text/javascript">  
    var str = "Javascript is an interesting scripting language";  
    var re = new RegExp( "script", "g" );  
  
    re.test(str);
```

```

    document.write("Test 1 - Current Index: " + re.lastIndex);

    re.test(str);
    document.write("<br />Test 2 - Current Index: " + re.lastIndex);

</script>
</body>
</html>

```

Output

```

Test 1 - Current Index: 10
Test 2 - Current Index: 35

```

multiline

multiline is a read-only boolean property of RegExp objects. It specifies whether a particular regular expression performs multiline matching, i.e., whether it was created with the "m" attribute.

Syntax

Its syntax is as follows:

```
RegExpObject.multiline
```

Return Value

Returns "TRUE" if the "m" modifier is set, "FALSE" otherwise.

Example

Try the following example program.

```

<html>
<head>
<title>JavaScript RegExp multiline Property</title>
</head>
<body>
<script type="text/javascript">
    var re = new RegExp( "string" );

```

```
if ( re.multiline ){
    document.write("Test1-multiline property is set");
}else{
    document.write("Test1-multiline property is not set");
}
re = new RegExp( "string", "m" );
if ( re.multiline ){
    document.write("<br/>Test2-multiline property is set");
}else{
    document.write("<br/>Test2-multiline property is not set");
}
</script>
</body>
</html>
```

Output

```
Test1-multiline property is not set
Test2-multiline property is set
```

source

source is a read-only string property of RegExp objects. It contains the text of the RegExp pattern. This text does not include the delimiting slashes used in regular-expression literals, and it does not include the "g", "i", and "m" attributes.

Syntax

Its syntax is as follows:

```
RegExpObject.source
```

Return Value

Returns the text used for pattern matching.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp source Property</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    re.test(str);
    document.write("The regular expression is : " + re.source);
</script>
</body>
</html>
```

Output

```
The regular expression is : script
```

RegExp Methods

Here is a list of the methods associated with RegExp along with their description.

Method	Description
exec()	Executes a search for a match in its string parameter.
test()	Tests for a match in its string parameter.
toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
toString()	Returns a string representing the specified object.

In the following sections, we will have a few examples to demonstrate the usage of RegExp methods.

exec()

The exec method searches string for text that matches regexp. If it finds a match, it returns an array of results; otherwise, it returns null.

Syntax

Its syntax is as follows:

```
RegExpObject.exec( string );
```

Parameter Details

string : The string to be searched.

Return Value

Returns the matched text if a match is found, and null if not.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp exec Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.exec(str);
    document.write("Test 1 - returned value : " + result);

    re = new RegExp( "pushing", "g" );
    var result = re.exec(str);
    document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```

Output

```
Test 1 - returned value : script
Test 2 - returned value : null
```

test()

The test method searches string for text that matches regexp. If it finds a match, it returns true; otherwise, it returns false.

Syntax

Its syntax is as follows:

```
RegExpObject.test( string );
```

Parameter Details

string : The string to be searched.

Return Value

Returns the matched text if a match is found, and null if not.

Example

Try the following example program.

```
<html>
<head>
<title>JavaScript RegExp test Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.test(str);
    document.write("Test 1 - returned value : " + result);

    re = new RegExp( "pushing", "g" );
    var result = re.test(str);
```

```

    document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>

```

Output

```

Test 1 - returned value : true
Test 2 - returned value : false

```

toSource ()

The toSource method string represents the source code of the object. This method does not work with all the browsers.

Syntax

Its syntax is as follows:

```

RegExpObject.toSource ( string );

```

Return Value

Returns the string representing the source code of the object.

Example

Try the following example program.

```

<html>
<head>
<title>JavaScript RegExp toSource Method</title>
</head>
<body>
<script type="text/javascript">
    var str = "Javascript is an interesting scripting language";
    var re = new RegExp( "script", "g" );

    var result = re.toSource(str);
    document.write("Test 1 - returned value : " + result);

```

```
re = new RegExp( "/", "g" );  
var result = re.toSource(str);  
document.write("<br />Test 2 - returned value : " + result);  
</script>  
</body>  
</html>
```

Output

```
Test 1 - returned value : /script/g  
Test 2 - returned value : /\//g
```

toString()

The `toString` method returns a string representation of a regular expression in the form of a regular-expression literal.

Syntax

Its syntax is as follows:

```
RegExpObject.toString ( );
```

Return Value

Returns the string representing of a regular expression.

Example

Try the following example program.

```
<html>  
<head>  
<title>JavaScript RegExp toString Method</title>  
</head>  
<body>  
<script type="text/javascript">  
    var str = "Javascript is an interesting scripting language";  
    var re = new RegExp( "script", "g" );  
  
    var result = re.toString(str);
```

```
document.write("Test 1 - returned value : " + result);

re = new RegExp( "/", "g" );
var result = re.toString(str);
document.write("<br />Test 2 - returned value : " + result);
</script>
</body>
</html>
```

Output

```
Test 1 - returned value : /script/g
Test 2 - returned value : /\//g
```

28. DOM

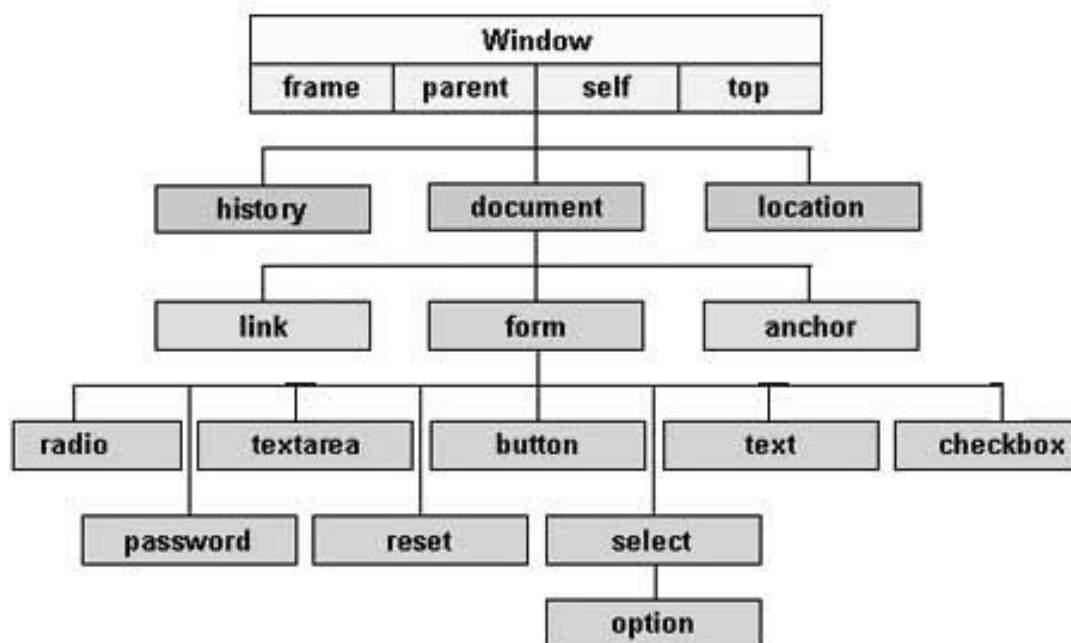
Every web page resides inside a browser window which can be considered as an object.

A Document object represents the HTML document that is displayed in that window. The Document object has various properties that refer to other objects which allow access to and modification of document content.

The way a document content is accessed and modified is called the Document Object Model, or DOM. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- Window object: Top of the hierarchy. It is the outmost element of the object hierarchy.
- Document object: Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- Form object: Everything enclosed in the <form>...</form> tags sets the form object.
- Form control elements: The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

Here is a simple hierarchy of a few important objects:



There are several DOMs in existence. The following sections explain each of these DOMs in detail and describe how you can use them to access and modify document content.

- The Legacy DOM : This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.
- The W3C DOM : This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.
- The IE4 DOM : This document object model was introduced in Version 4 of Microsoft's Internet Explorer browser. IE 5 and later versions include support for most basic W3C DOM features.

The Legacy DOM

This is the model which was introduced in early versions of JavaScript language. It is well supported by all browsers, but allows access only to certain key portions of documents, such as forms, form elements, and images.

This model provides several read-only properties, such as title, URL, and lastModified provide information about the document as a whole. Apart from that, there are various methods provided by this model which can be used to set and get document property values.

Document Properties in Legacy DOM

Here is a list of the document properties which can be accessed using Legacy DOM.

S.No	Property and Description
1	alinkColor Deprecated - A string that specifies the color of activated links. Ex : document.alinkColor
2	anchors[] An array of Anchor objects, one for each anchor that appears in the document

	Ex : document.anchors[0], document.anchors[1] and so on
3	<p>applets[]</p> <p>An array of Applet objects, one for each applet that appears in the document</p> <p>Ex: document.applets[0], document.applets[1] and so on</p>
4	<p>bgColor</p> <p>Deprecated - A string that specifies the background color of the document.</p> <p>Ex: document.bgColor</p>
5	<p>Cookie</p> <p>A string valued property with special behavior that allows the cookies associated with this document to be queried and set.</p> <p>Ex: document.cookie</p>
6	<p>Domain</p> <p>A string that specifies the Internet domain the document is from. Used for security purpose.</p> <p>Ex: document.domain</p>
7	<p>embeds[]</p> <p>An array of objects that represent data embedded in the document with the <embed> tag. A synonym for plugins []. Some plugins and ActiveX controls can be controlled with JavaScript code.</p> <p>Ex: document.embeds[0], document.embeds[1] and so on</p>
8	<p>fgColor</p> <p>A string that specifies the default text color for the document</p> <p>Ex: document.fgColor</p>

9	<p><code>forms[]</code></p> <p>An array of Form objects, one for each HTML form that appears in the document.</p> <p>Ex : <code>document.forms[0]</code>, <code>document.forms[1]</code> and so on</p>
10	<p><code>images[]</code></p> <p>An array of Image objects, one for each image that is embedded in the document with the HTML <code></code> tag.</p> <p>Ex : <code>document.images[0]</code>, <code>document.images[1]</code> and so on</p>
11	<p><code>lastModified</code></p> <p>A read-only string that specifies the date of the most recent change to the document</p> <p>Ex : <code>document.lastModified</code></p>
12	<p><code>linkColor</code></p> <p>Deprecated - A string that specifies the color of unvisited links</p> <p>Ex : <code>document.linkColor</code></p>
13	<p><code>links[]</code></p> <p>It is a document link array.</p> <p>Ex : <code>document.links[0]</code>, <code>document.links[1]</code> and so on</p>
14	<p><code>Location</code></p> <p>The URL of the document. Deprecated in favor of the <code>URL</code> property.</p> <p>Ex : <code>document.location</code></p>
15	<p><code>plugins[]</code></p> <p>A synonym for the <code>embeds[]</code></p>

	Ex : document.plugins[0], document.plugins[1] and so on
16	<p>Referrer</p> <p>A read-only string that contains the URL of the document, if any, from which the current document was linked.</p> <p>Ex : document.referrer</p>
17	<p>Title</p> <p>The text contents of the <title> tag.</p> <p>Ex : document.title</p>
18	<p>URL</p> <p>A read-only string that specifies the URL of the document.</p> <p>Ex : document.URL</p>
19	<p>vlinkColor</p> <p>Deprecated - A string that specifies the color of visited links.</p> <p>Ex : document.vlinkColor</p>

Document Methods in Legacy DOM

Here is a list of methods supported by Legacy DOM .

S.No	Property and Description
1	<p>clear()</p> <p>Deprecated - Erases the contents of the document and returns nothing.</p> <p>Ex : document.clear()</p>
2	<p>close()</p> <p>Closes a document stream opened with the open() method and returns nothing.</p>

	Ex : document.close()
3	<p>open()</p> <p>Deletes existing document content and opens a stream to which new document contents may be written. Returns nothing.</p> <p>Ex : document.open()</p>
4	<p>write(value, ...)</p> <p>Inserts the specified string or strings into the document currently being parsed or appends to document opened with open(). Returns nothing.</p> <p>Ex : document.write(value, ...)</p>
5	<p>writeln(value, ...)</p> <p>Identical to write(), except that it appends a newline character to the output. Returns nothing.</p> <p>Ex : document.writeln(value, ...)</p>

Example

We can locate any HTML element within any HTML document using HTML DOM. For instance, if a web document contains a `form` element, then using JavaScript, we can refer to it as `document.forms[0]`. If your Web document includes two `form` elements, the first form is referred to as `document.forms[0]` and the second as `document.forms[1]`.

Using the hierarchy and properties given above, we can access the first form element using `document.forms[0].elements[0]` and so on.

Here is an example to access document properties using Legacy DOM method.

```
<html>
<head>
<title> Document Title </title>
<script type="text/javascript">
<!--
function myFunc()
```

```
{  
    var ret = document.title;  
    alert("Document Title : " + ret );  
  
    var ret = document.URL;  
    alert("Document URL : " + ret );  
  
    var ret = document.forms[0];  
    alert("Document First Form : " + ret );  
  
    var ret = document.forms[0].elements[1];  
    alert("Second element : " + ret );  
  
}  
//-->  
</script>  
</head>  
<body>  
<h1 id="title">This is main title</h1>  
<p>Click the following to see the result:</p>  
  
<form name="FirstForm">  
<input type="button" value="Click Me" onclick="myFunc();" />  
<input type="button" value="Cancel">  
</form>  
  
<form name="SecondForm">  
<input type="button" value="Don't ClickMe"/>  
</form>  
  
</body>  
</html>
```

Output

This is main title

Click the following to see the result:

NOTE: This example returns objects for forms and elements and we would have to access their values by using those object properties which are not discussed in this tutorial.

The W3C DOM

This document object model allows access and modification of all document content and is standardized by the World Wide Web Consortium (W3C). This model is supported by almost all the modern browsers.

The W3C DOM standardizes most of the features of the legacy DOM and adds new ones as well. In addition to supporting forms[], images[], and other array properties of the Document object, it defines methods that allow scripts to access and manipulate any document element and not just special -purpose elements like forms and images.

Document Properties in W3C DOM

This model supports all the properties available in Legacy DOM. Additionally, here is a list of document properties which can be accessed using W3C DOM.

S.No	Property and Description
1	<p>Body</p> <p>A reference to the Element object that represents the <body> tag of this document.</p> <p>Ex : document.body</p>
2	<p>defaultView</p> <p>It is a read-only property and represents the window in which the document is displayed.</p>

	Ex : document.defaultView
3	documentElement A read-only reference to the <html> tag of the document. Ex : document.documentElement8/31/2008
4	Implementation It is a read-only property and represents the DOMImplementation object that represents the implementation that created this document. Ex : document.implementation

Document Methods in W3C DOM

This model supports all the methods available in Legacy DOM. Additionally, here is a list of methods supported by W3C DOM.

S.No	Property and Description
1	createAttribute(name) Returns a newly-created Attr node with the specified name. Ex : document.createAttribute(name)
2	createComment(text) Creates and returns a new Comment node containing the specified text. Ex : document.createComment(text)
3	createDocumentFragment() Creates and returns an empty DocumentFragment node. Ex : document.createDocumentFragment()

4	<p><code>createElement(tagName)</code></p> <p>Creates and returns a new Element node with the specified tag name.</p> <p>Ex: <code>document.createElement(tagName)</code></p>
---	---

Manual Animation

So let's implement one simple animation using DOM object properties and JavaScript functions as follows. The following list contains different DOM methods.

- We are using the JavaScript function `getElementById ()` to get a DOM object and then assigning it to a global variable `imgObj`.
- We have defined an initialization function `init ()` to initialize `imgObj` where we have set its position and left attributes.
- We are calling initialization function at the time of window load.
- Finally, we are calling `moveRight ()` function to increase the left distance by 10 pixels. You could also set it to a negative value to move it to the left side.

Example

Try the following example.

```
<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
function init(){
    imgObj = document.getElementById('myImage');
    imgObj.style.position= 'relative';
    imgObj.style.left = '0px';
}
function moveRight(){
    imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
}
window.onload =init;
//-->
</script>
</head>
<body>
```



```

<form>

<p>Click button below to move the image to right</p>
<input type="button" value="Click Me" onclick="moveRight();" />
</form>
</body>
</html>

```

Output

It is not possible to show animation in this tutorial. But you can [Try it here.](#)

Automated Animation

In the above example , we saw how an image moves to right with every click. We can automate this process by using the JavaScript function `setTimeout ()` as follows.

Here we have add ed more methods . So let's see what is new here:

- The `moveRight ()` function is calling `setTimeout ()` function to set the position of *imgObj*.
- We have added a new function `stop()` to clear the timer set by `setTimeout()` function and to set the object at its initial position.

Example

Try the following example code.

```

<html>
<head>
<title>JavaScript Animation</title>
<script type="text/javascript">
<!--
var imgObj = null;
var animate ;
function init(){
    imgObj = document.getElementById('myImage');
    imgObj.style.position= 'relative';
    imgObj.style.left = '0px';

```

```

}
function moveRight(){
    imgObj.style.left = parseInt(imgObj.style.left) + 10 + 'px';
    animate = setTimeout(moveRight,20); // call moveRight in 20msec
}
function stop(){
    clearTimeout(animate);
    imgObj.style.left = '0px';
}
window.onload =init;
//-->
</script>
</head>
<body>
<form>

<p>Click the buttons below to handle animation</p>
<input type="button" value="Start" onclick="moveRight();" />
<input type="button" value="Stop" onclick="stop();" />
</form>
</body>
</html>

```

It is not possible to show animation in this tutorial. But you can [Try it here.](#)

Rollover with a Mouse Event

Here is a simple example showing image rollover with a mouse event .

Let's see what we are using in the following example:

- At the time of loading this page, the `if()` statement checks for the existence of the image object. If the image object is unavailable, this block will not be executed.
- The `Image ()` constructor creates and preloads a new image object called `image1` .

- The src property is assigned the name of the external image file called /images/html.gif.
- Similarly, we have created image2 object and assigned /images/http.gif in this object.
- The # (hash mark) disables the link so that the browser does not try to go to a URL when clicked. This link is an image.
- The onMouseOver event handler is triggered when the user's mouse moves onto the link, and the onMouseOut event handler is triggered when the user's mouse moves away from the link (image).
- When the mouse moves over the image, the HTTP image changes from the first image to the second one. When the mouse is moved away from the image, the original image is displayed.
- When the mouse is moved away from the link, the initial image html.gif will reappear on the screen.

```
<html>
<head>
<title>Rollover with a Mouse Events</title>
<script type="text/javascript">
<!--
if(document.images){
    var image1 = new Image();      // Preload an image
    image1.src = "/images/html.gif";
    var image2 = new Image();      // Preload second image
    image2.src = "/images/http.gif";
}
//-->
</script>
</head>
<body>
<p>Move your mouse over the image to see the result</p>
<a href="#" onMouseOver="document.myImage.src=image2.src;"
    onMouseOut="document.myImage.src=image1.src;">

</a>
</body>
```

```
</html>
```

It is not possible to show animation in this tutorial. But you can [Try it here.](#)

32. MULTIMEDIA

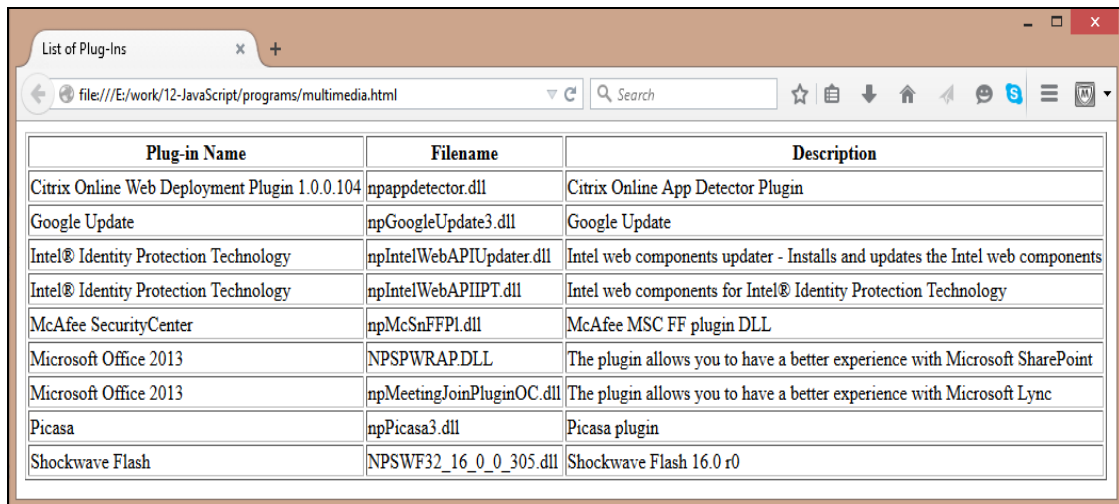
The JavaScript navigator object includes a child object called plugins . This object is an array, with one entry for each plug -in installed on the browser. The navigator.plugins object is supported only by Netscape, Firefox, and Mozilla only.

Example

Here is an example that shows how to list down all the plug-on installed in your browser:

```
<html>
<head>
<title>List of Plug-Ins</title>
</head>
<body>
<table border="1">
<tr><th>Plug-in Name</th><th>Filename</th><th>Description</th></tr>
<script LANGUAGE="JavaScript" type="text/javascript">
for (i=0; i<navigator.plugins.length; i++) {
    document.write("<tr><td>");
    document.write(navigator.plugins[i].name);
    document.write("</td><td>");
    document.write(navigator.plugins[i].filename);
    document.write("</td><td>");
    document.write(navigator.plugins[i].description);
    document.write("</td></tr>");
}
</script>
</table>
</body>
</html>
```

Output



The screenshot shows a web browser window with the title 'List of Plug-Ins'. The address bar shows the file path 'file:///E:/work/12-Javascript/programs/multimedia.html'. The browser displays a table with three columns: 'Plug-in Name', 'Filename', and 'Description'. The table lists various installed plug-ins, including Citrix Online Web Deployment Plugin, Google Update, Intel Identity Protection Technology, McAfee SecurityCenter, Microsoft Office 2013, Picasa, and Shockwave Flash.

Plug-in Name	Filename	Description
Citrix Online Web Deployment Plugin 1.0.0.104	npappdetector.dll	Citrix Online App Detector Plugin
Google Update	npGoogleUpdate3.dll	Google Update
Intel® Identity Protection Technology	npIntelWebAPIUpdater.dll	Intel web components updater - Installs and updates the Intel web components
Intel® Identity Protection Technology	npIntelWebAPIIPT.dll	Intel web components for Intel® Identity Protection Technology
McAfee SecurityCenter	npMcSnFFPI.dll	McAfee MSC FF plugin DLL
Microsoft Office 2013	NPSPWRAP.DLL	The plugin allows you to have a better experience with Microsoft SharePoint
Microsoft Office 2013	npMeetingJoinPluginOC.dll	The plugin allows you to have a better experience with Microsoft Lync
Picasa	npPicasa3.dll	Picasa plugin
Shockwave Flash	NPSWF32_16_0_0_305.dll	Shockwave Flash 16.0 r0

Checking for Plug-Ins

Each plug-in has an entry in the array. Each entry has the following properties:

- name - is the name of the plug-in.
- filename - is the executable file that was loaded to install the plug-in.
- description - is a description of the plug-in, supplied by the developer.
- mimeTypes - is an array with one entry for each MIME type supported by the plug-in.

You can use these properties in a script to find out the installed plug-ins, and then using JavaScript, you can play appropriate multimedia file. Take a look at the following example.

```
<html>
<head>
<title>Using Plug-Ins</title>
</head>
<body>
<script language="JavaScript" type="text/javascript">
media = navigator.mimeTypes["video/quicktime"];
if (media){
    document.write("<embed src='quick.mov' height=100 width=100>");
}
else{
```

```
document.write("<img src='quick.gif' height=100 width=100>");  
}  
</script>  
</body>  
</html>
```

NOTE: Here we are using HTML <embed> tag to embed a multimedia file.

Controlling Multimedia

Let us take a real example which works in almost all the browsers.

```
<html>  
<head>  
<title>Using Embedded Object</title>  
<script type="text/javascript">  
<!--  
function play()  
{  
    if (!document.demo.IsPlaying()){  
        document.demo.Play();  
    }  
}  
function stop()  
{  
    if (document.demo.IsPlaying()){  
        document.demo.StopPlay();  
    }  
}  
function rewind()  
    if (document.demo.IsPlaying()){  
        document.demo.StopPlay();  
    }  
    document.demo.Rewind();  
}
```

```
//-->
</script>
</head>
<body>
<embed id="demo" name="demo"
      src="http://www.amrood.com/games/kumite.swf"
      width="318" height="300" play="false" loop="false"
      pluginspage="http://www.macromedia.com/go/getflashplayer"
      swliveconnect="true">
</embed>
<form name="form" id="form" action="#" method="get">
<input type="button" value="Start" onclick="play();" />
<input type="button" value="Stop" onclick="stop();" />
<input type="button" value="Rewind" onclick="rewind();" />
</form>
</body>
</html>
```

If you are using Mozilla, Firefox or Netscape , then [Try it yourself](#).

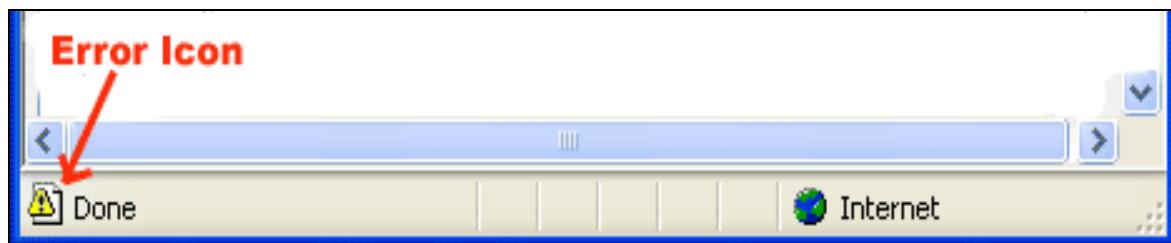
33. DEBUGGING

Every now and then, developers commit mistakes while coding. A mistake in a program or a script is referred to as a bug .

The process of finding and fixing bugs is called debugging and is a normal part of the development process. This section covers tools and techniques that can help you with debugging tasks.

Error Messages in IE

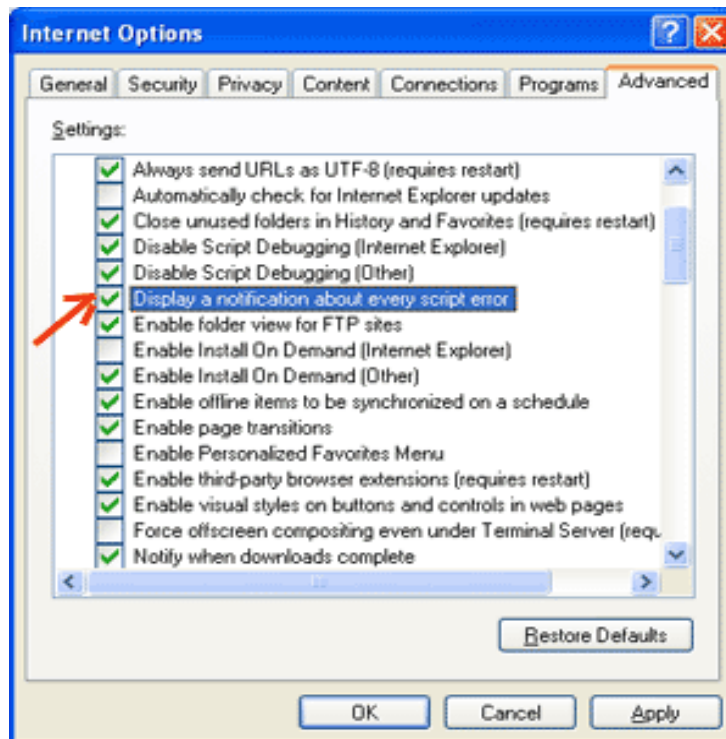
The most basic way to track down errors is by turning on error information in your browser. By default, Internet Explorer shows an error icon in the status bar when an error occurs on the page.



Double -clicking this icon takes you to a dialog box showing information about the specific error that occurred.

Since this icon is easy to overlook, Internet Explorer gives you the option to automatically show the Error dialog box whenever an error occurs.

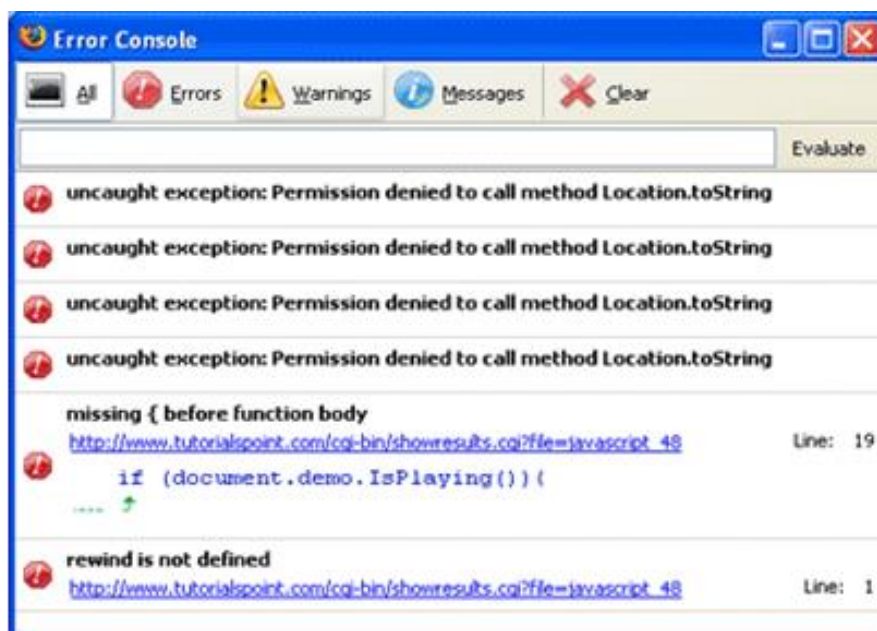
To enable this option, select Tools --> Internet Options --> Advanced tab and then finally check the "Display a Notification about Every Script Error" box option as shown below.



Error Messages in Firefox or Mozilla

Other browsers like Firefox, Netscape, and Mozilla send error messages to a special window called the JavaScript Console or Error Console. To view the console, select Tools --> Error Console or Web Development.

Unfortunately, since these browsers give no visual indication when an error occurs, you must keep the Console open and watch for errors as your script executes.



Error Notifications

Error notifications that show up on Console or through Internet Explorer dialog boxes are the result of both syntax and runtime errors. These error notification include the line number at which the error occurred.

If you are using Firefox , then you can click on the error available in the error console to go to the exact line in the script having error.

How to Debug a Script

There are various ways to debug your JavaScript:

Use a JavaScript Validator

One way to check your JavaScript code for strange bugs is to run it through a program that checks it to make sure it is valid and that it follows the official syntax rules of the language. These programs are called validating parsers or just validators for short, and often come with commercial HTML and JavaScript editors.

The most convenient validator for JavaScript is Douglas Crockford's JavaScript Lint, which is available for free at [Douglas Crockford's JavaScript Lint](http://jshint.com/).

Simply visit that web page, paste your JavaScript (Only JavaScript) code into the text area provided, and click the jslint button. This program will parse through your JavaScript code, ensuring that all the variable and function definitions follow the correct syntax. It will also check JavaScript statements, such as if and while , to ensure they too follow the correct format

Add Debugging Code to Your Programs

You can use the alert () or document.write () methods in your program to debug your code. For example, you might write something as follows:

```
var debugging = true;
var whichImage = "widget";
if( debugging )
    alert( "Calls swapImage() with argument: " + whichImage );
var swapStatus = swapImage( whichImage );
if( debugging )
    alert( "Exits swapImage() with swapStatus=" + swapStatus );
```

By examining the content and order of the alert () as they appear, you can examine the health of your program very easily.

Use a JavaScript Debugger

A debugger is an application that places all aspects of script execution under the control of the programmer. Debuggers provide fine -grained control over the state of the script through an interface that allows you to examine and set values as well as control the flow of execution.

Once a script has been loaded into a debugger, it can be run one line at a time or instructed to halt at certain breakpoints. Once execution is halted, the programmer can examine the state of the script and its variables in order to determine if something is amiss. You can also watch variables for changes in their values.

The latest version of the Mozilla JavaScript Debugger (code -named Venkman) for both Mozilla and Netscape browsers can be downloaded at <http://www.hacksrus.com/~ginda/venkman>.

Useful Tips for Developers

You can keep the following tips in mind to reduce the number of errors in your scripts and simplify the debugging process:

- Use plenty of comments . Comments enable you to explain why you wrote the script the way you did and to explain particularly difficult sections of code.
- Always use indentation to make your code easy to read. Indenting statements also makes it easier for you to match up beginning and ending tags, curly braces, and other HTML and script elements.
- Write modular code . Whenever possible, group your statements into functions. Functions let you group related statements, and test and reuse portions of code with minimal effort.
- Be consistent in the way you name your variables and functions. Try using names that are long enough to be meaningful and that describe the contents of the variable or the purpose of the function.
- Use consistent syntax when naming variables and functions. In other words, keep them all lowercase or all uppercase; if you prefer Camel -Back notation, use it consistently.
- Test long scripts in a modular fashion. In other words, do not try to write the entire script before testing any portion of it. Write a piece and get it to work before adding the next portion of code.
- Use descriptive variable and function names and avoid using single -character names.

- Watch your quotation marks . Remember that quotation marks are used in pairs around strings and that both quotation marks must be of the same style (either single or double).
- Watch your equal signs . You should not use a single = for comparison purpose.
- Declare variables explicitly using the var keyword.

34. IMAGE MAP

You can use JavaScript to create client-side image map. Client-side image maps are enabled by the `usemap` attribute for the `` tag and defined by special `<map>` and `<area>` extension tags.

The image that is going to form the map is inserted into the page using the `` element as normal, except that it carries an extra attribute called `usemap`. The value of the `usemap` attribute is the value of the `name` attribute on the `<map>` element, which you are about to meet, preceded by a pound or hash sign.

The `<map>` element actually creates the map for the image and usually follows directly after the `` element. It acts as a container for the `<area />` elements that actually define the clickable hotspots. The `<map>` element carries only one attribute, the `name` attribute, which is the name that identifies the map. This is how the `` element knows which `<map>` element to use.

The `<area>` element specifies the shape and the coordinates that define the boundaries of each clickable hotspot.

The following code combines imagemaps and JavaScript to produce a message in a text box when the mouse is moved over different parts of an image.

```
<html>
<head>
<title>Using JavaScript Image Map</title>
<script type="text/javascript">
    <!--
        function showTutorial(name){
            document.myform.stage.value = name
        }
    //-->
</script>
</head>
<body>
<form name="myform">
    <input type="text" name="stage" size="20" />
</form>
```

```
<!-- Create Mappings -->

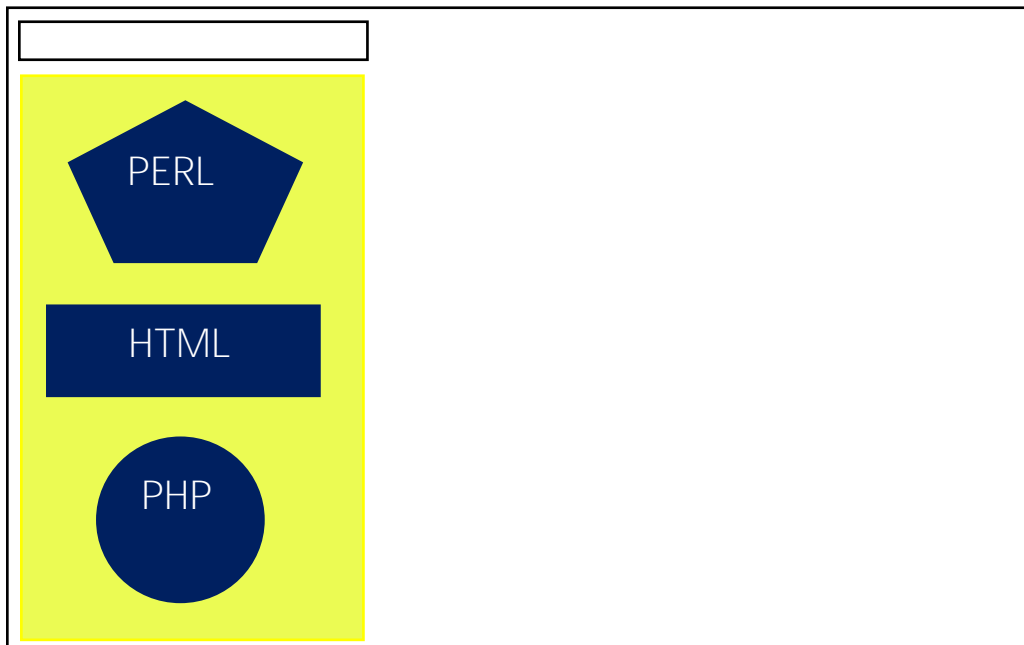

<map name="tutorials">
  <area shape="poly"
        coords="74,0,113,29,98,72,52,72,38,27"
        href="/perl/index.htm" alt="Perl Tutorial"
        target="_self"
        onMouseOver="showTutorial('perl')"
        onMouseOut="showTutorial('')"/>

  <area shape="rect"
        coords="22,83,126,125"
        href="/html/index.htm" alt="HTML Tutorial"
        target="_self"
        onMouseOver="showTutorial('html')"
        onMouseOut="showTutorial('')"/>

  <area shape="circle"
        coords="73,168,32"
        href="/php/index.htm" alt="PHP Tutorial"
        target="_self"
        onMouseOver="showTutorial('php')"
        onMouseOut="showTutorial('')"/>
</map>
</body>
</html>
```

Output

You can feel the map concept by placing the mouse cursor on the image object.



35. BROWSERS

It is important to understand the differences between different browsers in order to handle each in the way it is expected. So it is important to know which browser your web page is running in.

To get information about the browser your webpage is currently running in, use the built-in navigator object.

Navigator Properties

There are several Navigator related properties that you can use in your Web page. The following is a list of the names and descriptions of each .

S.No	Property and Description
1	<code>appName</code> This property is a string that contains the code name of the browser, Netscape for Netscape and Microsoft Internet Explorer for Internet Explorer.
2	<code>appVersion</code> This property is a string that contains the version of the browser as well as other useful information such as its language and compatibility.
3	<code>language</code> This property contains the two-letter abbreviation for the language that is used by the browser. Netscape only.
4	<code>mimTypes[]</code> This property is an array that contains all MIME types supported by the client. Netscape only.
5	<code>platform[]</code> This property is a string that contains the platform for which the browser was compiled."Win32" for 32-bit Windows operating systems

6	<p><code>plugins[]</code></p> <p>This property is an array containing all the plug -ins that have been installed on the client. Netscape only.</p>
7	<p><code>userAgent[]</code></p> <p>This property is a string that contains the code name and version of the browser. This value is sent to the originating server to identify the client.</p>

Navigator Methods

There are several Navigator -specific methods. Here is a list of their names and descriptions.

S.No	Method and Description
1	<p><code>javaEnabled()</code></p> <p>This method determines if JavaScript is enabled in the client. If JavaScript is enabled, this method returns true; otherwise, it returns false.</p>
2	<p><code>plugings.refresh</code></p> <p>This method makes newly installed plug -ins available and populates the plugins array with all new plug -in names. Netscape only.</p>
3	<p><code>preference(name,value)</code></p> <p>This method allows a signed script to get and set some Netscape preferences. If the second parameter is omitted, this method will return the value of the specified preference; otherwise, it sets the value. Netscape only.</p>
4	<p><code>taintEnabled()</code></p> <p>This method returns true if data tainting is enabled ; false otherwise.</p>

Browser Detection

There is a simple JavaScript which can be used to find out the name of a browser and then accordingly an HTML page can be served to the user.

```
<html>
<head>
<title>Browser Detection Example</title>
</head>
<body>
<script type="text/javascript">
<!--
var userAgent    = navigator.userAgent;
var opera        = (userAgent.indexOf('Opera') != -1);
var ie           = (userAgent.indexOf('MSIE') != -1);
var gecko        = (userAgent.indexOf('Gecko') != -1);
var netscape     = (userAgent.indexOf('Mozilla') != -1);
var version      = navigator.appVersion;

if (opera){
    document.write("Opera based browser");
    // Keep your opera specific URL here.
}else if (gecko){
    document.write("Mozilla based browser");
    // Keep your gecko specific URL here.
}else if (ie){
    document.write("IE based browser");
    // Keep your IE specific URL here.
}else if (netscape){
    document.write("Netscape based browser");
    // Keep your Netscape specific URL here.
}else{
    document.write("Unknown browser");
}
// You can include version to along with any above condition.
```

```
document.write("<br /> Browser version info : " + version );  
//-->  
</script>  
</body>  
</html>
```

Output

```
Mozilla based browser  
Browser version info : 5.0  
(Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)  
Chrome/41.0.2272.101 Safari/537.36
```