**Course Web Programming Unit III**

***Contents***

*Java Script: Introduction, Client-Side JavaScript, Server-Side JavaScript, JavaScript Objects, JavaScript Security, Operators: Assignment Operators, Comparison Operators, Arithmetic Operators, Logical Operators, Short-Circuit Evaluation, String Operators, Special Operators, (Conditional operator), ,(Comma operator), delete, new, this, void Statements : Break, comment, continue, delete, do … while, export, for, for…in, function, if…else, import, labelled, return, switch, var, while, with, Core JavaScript (Properties and Methods of Each) : Array, Boolean, Date, Function, Math, Number, Object, String, regExp Document and its associated objects : document, Link, Area, Anchor, Image, Applet, Layer Events and Event Handlers : General Information about Events, Defining Event Handlers and events*

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](http://www.ecma-international.org/publications/index.html) defined a standard version of the core JavaScript language.

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

Client-side JavaScript

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

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

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

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

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

Advantages of JavaScript

The merits of using JavaScript are −

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

Limitations of JavaScript

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

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

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

JavaScript Development Tools

One of major strengths of JavaScript is that it does not require expensive development tools. You can start with a simple text editor such as Notepad. Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.

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

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

Where is JavaScript Today?

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

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

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

JavaScript Syntax

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

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

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

<script ...>

JavaScript code

</script>

The script tag takes two important attributes −

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

So your JavaScript segment will look like −

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

JavaScript code

</script>

Your First JavaScript Script

Let us take a sample example to print out "Hello World". We added an optional HTML comment that surrounds our JavaScript code. This is to save our code from a browser that does not support JavaScript. The comment ends with a "//-->". Here "//" signifies a comment in JavaScript, so we add that to prevent a browser from reading the end of the HTML comment as a piece of JavaScript code. Next, we call a function **document.write** which writes a string into our HTML document.

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

<html>

<body>

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

<!--

document.write("Hello World!")

//-->

</script>

</body>

</html>

This code will produce the following result −

Hello World!

Whitespace and Line Breaks

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

Semicolons are Optional

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

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

<!--

var1 = 10

var2 = 20

//-->

</script>

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

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

<!--

var1 = 10; var2 = 20;

//-->

</script>

**Note** − It is a good programming practice to use semicolons.

Case Sensitivity

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

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

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

Comments in JavaScript

JavaScript supports both C-style and C++-style comments, Thus −

* Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
* Any text between the characters /\* and \*/ is treated as a comment. This may span multiple lines.
* JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
* The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example

The following example shows how to use comments in JavaScript.

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

<!--

// This is a comment. It is similar to comments in C++

/\*

\* This is a multiline comment in JavaScript

\* It is very similar to comments in C Programming

\*/

//-->

</script>

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 simple steps to turn on or turn off JavaScript in your 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 **Scripting option.**
* Select *Enable* radio button under **Active scripting**.
* Finally click OK and come out

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

JavaScript in Firefox

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

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

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

JavaScript in Chrome

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

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

JavaScript in Opera

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

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

To disable JavaScript support in your 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

Placement in HTML FILE

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>

<input type="button" onclick="sayHello()" value="Say Hello" />

</body>

</html>

This code will produce the following results −

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 −

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 −

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")

}

# JavaScript - Variables

JavaScript Datatypes

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

JavaScript allows you to work with three primitive data types −

* **Numbers,** eg. 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** − JavaScript 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.

<html>

<body onload = checkscope();>

<script type = "text/javascript">

<!--

var myVar = "global"; // Declare a global variable

function checkscope( ) {

var myVar = "local"; // Declare a local variable

document.write(myVar);

}

//-->

</script>

</body>

</html>

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

# JavaScript - Operators

What is an operator?

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

* Arithmetic Operators
* Comparision Operators
* Logical (or Relational) Operators
* Assignment Operators
* Conditional (or ternary) Operators

Lets have a look on all operators one by one.

Arithmetic Operators

JavaScript supports the following arithmetic operators −

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

|  |  |
| --- | --- |
| **Sr.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>

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

</body>

</html>

Output

a + b = 43

a - b = 23

a / b = 3.3

a % b = 3

a + b + c = 43Test

++a = 35

--b = 8

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 −

|  |  |
| --- | --- |
| **Sr.No** | **Operator and Description** |
| 1 | = = **(Equal)**  Checks if the value of two operands are equal or not, if yes, then the condition becomes true.  **Ex:** (A == B) is not true. |
| 2 | **!= (Not Equal)**  Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.  **Ex:** (A != B) is true. |
| 3 | **> (Greater than)**  Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A > B) is not true. |
| 4 | **< (Less than)**  Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A < B) is true. |
| 5 | **>= (Greater than or Equal to)**  Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.  **Ex:** (A >= B) is not true. |
| 6 | **<= (Less than or Equal to)**  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.  **Ex:** (A <= B) is true. |

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>

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

</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 −

|  |  |
| --- | --- |
| **Sr.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 −

|  |  |
| --- | --- |
| **Sr.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 | **<< (Left Shift)**  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.  **Ex:** (A << 1) is 4. |
| 6 | **>> (Right Shift)**  Binary Right Shift Operator. The left operand’s value is moved right by the number of bits specified by the right operand.  **Ex:** (A >> 1) is 1. |
| 7 | **>>> (Right shift with Zero)**  This operator is just like the >> operator, except that the bits shifted in on the left are always zero.  **Ex:** (A >>> 1) is 1. |

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>

(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 −

|  |  |
| --- | --- |
| **Sr.No** | **Operator and Description** |
| 1 | **= (Simple Assignment )**  Assigns values from the right side operand to the left side operand  **Ex:** C = A + B will assign the value of A + B into C |
| 2 | **+= (Add and Assignment)**  It adds the right operand to the left operand and assigns the result to the left operand.  **Ex:** C += A is equivalent to C = C + A |
| 3 | **−= (Subtract and Assignment)**  It subtracts the right operand from the left operand and assigns the result to the left operand.  **Ex:** C -= A is equivalent to C = C - A |
| 4 | **\*= (Multiply and Assignment)**  It multiplies the right operand with the left operand and assigns the result to the left operand.  **Ex:** C \*= A is equivalent to C = C \* A |
| 5 | **/= (Divide and Assignment)**  It divides the left operand with the right operand and assigns the result to the left operand.  **Ex:** C /= A is equivalent to C = C / A |
| 6 | **%= (Modules and Assignment)**  It takes modulus using two operands and assigns the result to the left operand.  **Ex:** C %= A is equivalent to C = C % A |

**Note** − Same logic applies to Bitwise operators so they will become like <<=, >>=, >>=, &=, |= 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 Operator

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.

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

Example

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

<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = 20;

var linebreak = "<br />";

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

result = (a > b) ? 100 : 200;

document.write(result);

document.write(linebreak);

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

result = (a < b) ? 100 : 200;

document.write(result);

document.write(linebreak);

//-->

</script>

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

</body>

</html>

Output

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

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

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

typeof Operator

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

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

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

|  |  |
| --- | --- |
| **Type** | **String Returned by typeof** |
| Number | "number" |
| String | "string" |
| Boolean | "boolean" |
| Object | "object" |
| Function | "function" |
| Undefined | "undefined" |
| Null | "object" |

Example

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

<html>

<body>

<script type="text/javascript">

<!--

var a = 10;

var b = "String";

var linebreak = "<br />";

result = (typeof b == "string" ? "B is String" : "B is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);

result = (typeof a == "string" ? "A is String" : "A is Numeric");

document.write("Result => ");

document.write(result);

document.write(linebreak);

//-->

</script>

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

</body>

</html>

Output

Result => B is String

Result => A is Numeric

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

# JavaScript - if...else Statement

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

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** 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...

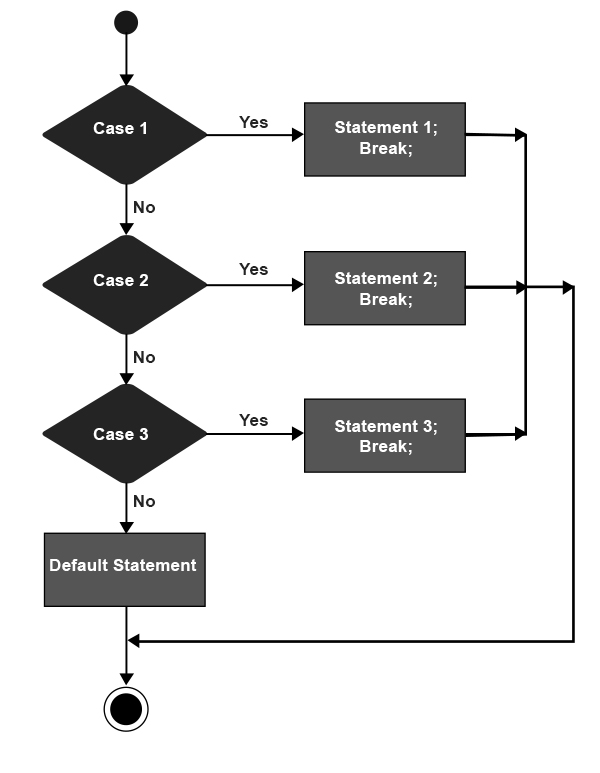
# JavaScript - Switch Case

You can use multiple **if...else…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...

# JavaScript - While Loops

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** − Don’t miss the semicolon 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...

# JavaScript - For Loop

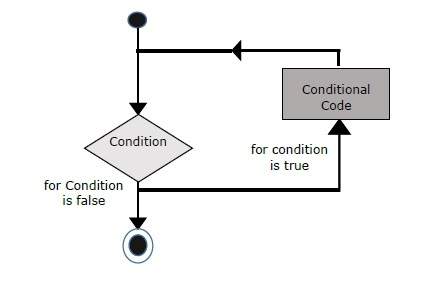
The '**for**' loop is the most compact form of looping. It includes the following three important parts −

* 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 is 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...

# JavaScript *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

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 implement ‘for-in’ loop. It prints the web browser’s **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

mimeTypes

plugins

javaEnabled

getStorageUpdates

getGamepads

webkitGetUserMedia

vibrate

getBattery

sendBeacon

registerProtocolHandler

unregisterProtocolHandler

Exiting from the loop!

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

# JavaScript - 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 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.

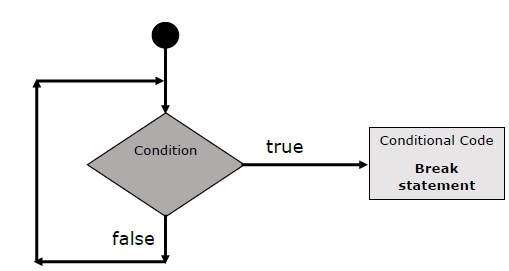
To handle all such situations, JavaScript provides **break** and **continue** statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

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 already have 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

<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!

# JavaScript - 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

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

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

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

* [JavaScript Nested Functions](https://www.tutorialspoint.com/javascript/javascript_nested_functions.htm)
* [JavaScript Function( ) Constructor](https://www.tutorialspoint.com/javascript/javascript_function_constructors.htm)
* [JavaScript Function Literals](https://www.tutorialspoint.com/javascript/javascript_function_literals.htm)

# JavaScript - 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](https://www.tutorialspoint.com/html/html_events_ref.htm). Here we will see a few examples to understand a 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() {

alert("Hello World")

}

//-->

</script>

</head>

<body>

<p>Click the following button and see result</p>

<form>

<input type="button" onclick="sayHello()" value="Say Hello" />

</form>

</body>

</html>

Output

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 your mouse out from that element. Try the following example.

<html>

<head>

<script type="text/javascript">

<!--

function over() {

document.write ("Mouse Over");

}

function out() {

document.write ("Mouse Out");

}

//-->

</script>

</head>

<body>

<p>Bring your mouse inside the division to see the result:</p>

<div onmouseover="over()" onmouseout="out()">

<h2> This is inside the division </h2>

</div>

</body>

</html>

Output

HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Value** | **Description** |
| Offline | script | Triggers when the document goes offline |
| Onabort | script | Triggers on an abort event |
| onafterprint | script | Triggers after the document is printed |
| onbeforeonload | script | Triggers before the document loads |
| onbeforeprint | script | Triggers before the document is printed |
| onblur | script | Triggers when the window loses focus |
| oncanplay | script | Triggers when media can start play, but might has to stop for buffering |
| oncanplaythrough | script | Triggers when media can be played to the end, without stopping for buffering |
| onchange | script | Triggers when an element changes |
| onclick | script | Triggers on a mouse click |
| oncontextmenu | script | Triggers when a context menu is triggered |
| ondblclick | script | Triggers on a mouse double-click |
| ondrag | script | Triggers when an element is dragged |
| ondragend | script | Triggers at the end of a drag operation |
| ondragenter | script | Triggers when an element has been dragged to a valid drop target |
| ondragleave | script | Triggers when an element is being dragged over a valid drop target |
| ondragover | script | Triggers at the start of a drag operation |
| ondragstart | script | Triggers at the start of a drag operation |
| ondrop | script | Triggers when dragged element is being dropped |
| ondurationchange | script | Triggers when the length of the media is changed |
| onemptied | script | Triggers when a media resource element suddenly becomes empty. |
| onended | script | Triggers when media has reach the end |
| onerror | script | Triggers when an error occur |
| onfocus | script | Triggers when the window gets focus |
| onformchange | script | Triggers when a form changes |
| onforminput | script | Triggers when a form gets user input |
| onhaschange | script | Triggers when the document has change |
| oninput | script | Triggers when an element gets user input |
| oninvalid | script | Triggers when an element is invalid |
| onkeydown | script | Triggers when a key is pressed |
| onkeypress | script | Triggers when a key is pressed and released |
| onkeyup | script | Triggers when a key is released |
| onload | script | Triggers when the document loads |
| onloadeddata | script | Triggers when media data is loaded |
| onloadedmetadata | script | Triggers when the duration and other media data of a media element is loaded |
| onloadstart | script | Triggers when the browser starts to load the media data |
| onmessage | script | Triggers when the message is triggered |
| onmousedown | script | Triggers when a mouse button is pressed |
| onmousemove | script | Triggers when the mouse pointer moves |
| onmouseout | script | Triggers when the mouse pointer moves out of an element |
| onmouseover | script | Triggers when the mouse pointer moves over an element |
| onmouseup | script | Triggers when a mouse button is released |
| onmousewheel | script | Triggers when the mouse wheel is being rotated |
| onoffline | script | Triggers when the document goes offline |
| onoine | script | Triggers when the document comes online |
| ononline | script | Triggers when the document comes online |
| onpagehide | script | Triggers when the window is hidden |
| onpageshow | script | Triggers when the window becomes visible |
| onpause | script | Triggers when media data is paused |
| onplay | script | Triggers when media data is going to start playing |
| onplaying | script | Triggers when media data has start playing |
| onpopstate | script | Triggers when the window's history changes |
| onprogress | script | Triggers when the browser is fetching the media data |
| onratechange | script | Triggers when the media data's playing rate has changed |
| onreadystatechange | script | Triggers when the ready-state changes |
| onredo | script | Triggers when the document performs a redo |
| onresize | script | Triggers when the window is resized |
| onscroll | script | Triggers when an element's scrollbar is being scrolled |
| onseeked | script | Triggers when a media element's seeking attribute is no longer true, and the seeking has ended |
| onseeking | script | Triggers when a media element's seeking attribute is true, and the seeking has begun |
| onselect | script | Triggers when an element is selected |
| onstalled | script | Triggers when there is an error in fetching media data |
| onstorage | script | Triggers when a document loads |
| onsubmit | script | Triggers when a form is submitted |
| onsuspend | script | Triggers when the browser has been fetching media data, but stopped before the entire media file was fetched |
| ontimeupdate | script | Triggers when media changes its playing position |
| onundo | script | Triggers when a document performs an undo |
| onunload | script | Triggers when the user leaves the document |
| onvolumechange | script | Triggers when media changes the volume, also when volume is set to "mute" |
| onwaiting | script | Triggers when media has stopped playing, but is expected to resume |

**void** is an important keyword in JavaScript which can be used as a unary operator that appears before its single operand, which may be of any type. This operator specifies an expression to be evaluated without returning a value.

Syntax

The syntax of **void** can be either of the following two −

<head>

<script type="text/javascript">

<!--

void func()

javascript:void func()

or:

void(func())

javascript:void(func())

//-->

</script>

</head>

Example 1

The most common use of this operator is in a client-side *javascript:* URL, where it allows you to evaluate an expression for its side-effects without the browser displaying the value of the evaluated expression.

Here the expression **alert ('Warning!!!')** is evaluated but it is not loaded back into the current document −

<html>

<head>

<script type="text/javascript">

<!--

//-->

</script>

</head>

<body>

<p>Click the following, This won't react at all...</p>

<a href="javascript:void(alert('Warning!!!'))">Click me!</a>

</body>

</html>

Click the following, This won't react at all...

[Click me!](javascript:void(alert('Warning!!!')))

# JavaScript - Objects Overview

JavaScript is an Object Oriented Programming (OOP) language. A programming language can be called object-oriented if it provides four basic capabilities to developers −

* **Encapsulation** − the capability to store related information, whether data or methods, together in an object.
* **Aggregation** − the capability to store one object inside another object.
* **Inheritance** − the capability of a class to rely upon another class (or number of classes) for some of its properties and methods.
* **Polymorphism** − the capability to write one function or method that works in a variety of different ways.

Objects are composed of attributes. If an attribute contains a function, it is considered to be a method of the object, otherwise the attribute is considered a property.

Object Properties

Object properties can be any of the three primitive data types, or any of the abstract data types, such as another object. Object properties are usually variables that are used internally in the object's methods, but can also be globally visible variables that are used throughout the page.

The syntax for adding a property to an object is −

objectName.objectProperty = propertyValue;

**For example** − The following code gets the document title using the **"title"** property of the **document** object.

var str = document.title;

Object Methods

Methods are the functions that let the object do something or let something be done to it. There is a small difference between a function and a method – at a function is a standalone unit of statements and a method is attached to an object and can be referenced by the **this** keyword.

Methods are useful for everything from displaying the contents of the object to the screen to performing complex mathematical operations on a group of local properties and parameters.

**For example** − Following is a simple example to show how to use the **write()** method of document object to write any content on the document.

document.write("This is test");

User-Defined Objects

All user-defined objects and built-in objects are descendants of an object called **Object**.

The new Operator

The **new** operator is used to create an instance of an object. To create an object, the **new** operator is followed by the constructor method.

In the following example, the constructor methods are Object(), Array(), and Date(). These constructors are built-in JavaScript functions.

var employee = new Object();

var books = new Array("C++", "Perl", "Java");

var day = new Date("August 15, 1947");

The Object() Constructor

A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

The variable contains a reference to the new object. The properties assigned to the object are not variables and are not defined with the **var** keyword.

Example 1

Try the following example; it demonstrates how to create an Object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

var book = new Object(); // Create the object

book.subject = "Perl"; // Assign properties to the object

book.author = "Mohtashim";

</script>

</head>

<body>

<script type="text/javascript">

document.write("Book name is : " + book.subject + "<br>");

document.write("Book author is : " + book.author + "<br>");

</script>

</body>

</html>

Output

Book name is : Perl

Book author is : Mohtashim

Example 2

This example demonstrates how to create an object with a User-Defined Function. Here **this** keyword is used to refer to the object that has been passed to a function.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

function book(title, author){

this.title = title;

this.author = author;

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

</script>

</body>

</html>

Output

Book title is : Perl

Book author is : Mohtashim

Defining Methods for an Object

The previous examples demonstrate how the constructor creates the object and assigns properties. But we need to complete the definition of an object by assigning methods to it.

Example

Try the following example; it shows how to add a function along with an object.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

this.price = amount;

}

function book(title, author){

this.title = title;

this.author = author;

this.addPrice = addPrice; // Assign that method as property.

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>

</html>

Output

Book title is : Perl

Book author is : Mohtashim

Book price is : 100

The 'with' Keyword

The **‘with’** keyword is used as a kind of shorthand for referencing an object's properties or methods.

The object specified as an argument to **with** becomes the default object for the duration of the block that follows. The properties and methods for the object can be used without naming the object.

Syntax

The syntax for with object is as follows −

with (object){

properties used without the object name and dot

}

Example

Try the following example.

<html>

<head>

<title>User-defined objects</title>

<script type="text/javascript">

// Define a function which will work as a method

function addPrice(amount){

with(this){

price = amount;

}

}

function book(title, author){

this.title = title;

this.author = author;

this.price = 0;

this.addPrice = addPrice; // Assign that method as property.

}

</script>

</head>

<body>

<script type="text/javascript">

var myBook = new book("Perl", "Mohtashim");

myBook.addPrice(100);

document.write("Book title is : " + myBook.title + "<br>");

document.write("Book author is : " + myBook.author + "<br>");

document.write("Book price is : " + myBook.price + "<br>");

</script>

</body>

</html>

Output

Book title is : Perl

Book author is : Mohtashim

Book price is : 100

JavaScript Native Objects

JavaScript has several built-in or native objects. These objects are accessible anywhere in your program and will work the same way in any browser running in any operating system.

Here is the list of all important JavaScript Native Objects −

* [JavaScript Number Object](https://www.tutorialspoint.com/javascript/javascript_number_object.htm)
* [JavaScript Boolean Object](https://www.tutorialspoint.com/javascript/javascript_boolean_object.htm)
* [JavaScript String Object](https://www.tutorialspoint.com/javascript/javascript_strings_object.htm)
* [JavaScript Array Object](https://www.tutorialspoint.com/javascript/javascript_arrays_object.htm)
* [JavaScript Date Object](https://www.tutorialspoint.com/javascript/javascript_date_object.htm)
* [JavaScript Math Object](https://www.tutorialspoint.com/javascript/javascript_math_object.htm)
* [JavaScript RegExp Object](https://www.tutorialspoint.com/javascript/javascript_regexp_object.htm)

# JavaScript - The Boolean Object

The **Boolean** object represents two values, either "true" or "false". If *value* parameter is omitted or is 0, -0, null, false, **NaN,** undefined, or the empty string (""), the object has an initial value of false.

Syntax

Use the following syntax to create a **boolean** object.

var val = new Boolean(value);

Boolean Properties

Here is a list of the properties of Boolean object −

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.tutorialspoint.com/javascript/boolean_constructor.htm) | Returns a reference to the Boolean function that created the object. |

# JavaScript - The Number Object

The **Number** object represents numerical date, either integers or floating-point numbers. In general, you do not need to worry about **Number** objects because the browser automatically converts number literals to instances of the number class.

Syntax

The syntax for creating a **number** object is as follows −

var val = new Number(number);

In the place of number, if you provide any non-number argument, then the argument cannot be converted into a number, it returns **NaN** (Not-a-Number).

Number Properties

Here is a list of each property and their description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [MAX\_VALUE](https://www.tutorialspoint.com/javascript/number_max_value.htm) | The largest possible value a number in JavaScript can have 1.7976931348623157E+308 |
| [MIN\_VALUE](https://www.tutorialspoint.com/javascript/number_min_value.htm) | The smallest possible value a number in JavaScript can have 5E-324 |
| [NaN](https://www.tutorialspoint.com/javascript/number_nan.htm) | Equal to a value that is not a number. |
| [NEGATIVE\_INFINITY](https://www.tutorialspoint.com/javascript/number_negative_infinity.htm) | A value that is less than MIN\_VALUE. |
| [POSITIVE\_INFINITY](https://www.tutorialspoint.com/javascript/number_positive_infinity.htm) | A value that is greater than MAX\_VALUE |
| [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm) | A static property of the Number object. Use the prototype property to assign new properties and methods to the Number object in the current document |
| [constructor](https://www.tutorialspoint.com/javascript/number_constructor.htm) | Returns the function that created this object's instance. By default this is the Number object. |

In the following sections, we will take a few examples to demonstrate the properties of Number.

Number Methods

The Number object contains only the default methods that are a part of every object's definition.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [toExponential()](https://www.tutorialspoint.com/javascript/number_toexponential.htm) | Forces a number to display in exponential notation, even if the number is in the range in which JavaScript normally uses standard notation. |
| [toFixed()](https://www.tutorialspoint.com/javascript/number_tofixed.htm) | Formats a number with a specific number of digits to the right of the decimal. |
| [toLocaleString()](https://www.tutorialspoint.com/javascript/number_tolocalestring.htm) | Returns a string value version of the current number in a format that may vary according to a browser's local settings. |
| [toPrecision()](https://www.tutorialspoint.com/javascript/number_toprecision.htm) | Defines how many total digits (including digits to the left and right of the decimal) to display of a number. |
| [toString()](https://www.tutorialspoint.com/javascript/number_tostring.htm) | Returns the string representation of the number's value. |
| [valueOf()](https://www.tutorialspoint.com/javascript/number_valueof.htm) | Returns the number's value. |

# JavaScript - The Strings Object

The **String** object lets you work with a series of characters; it wraps Javascript's string primitive data type with a number of helper methods.

As JavaScript automatically converts between string primitives and String objects, you can call any of the helper methods of the String object on a string primitive.

Syntax

Use the following syntax to create a String object −

var val = new String(string);

The **String** parameter is a series of characters that has been properly encoded.

String Properties

Here is a list of the properties of String object and their description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.tutorialspoint.com/javascript/string_constructor.htm) | Returns a reference to the String function that created the object. |
| [length](https://www.tutorialspoint.com/javascript/string_length.htm) | Returns the length of the string. |
| [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm) | The prototype property allows you to add properties and methods to an object. |

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

String Methods

Here is a list of the methods available in String object along with their description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [charAt()](https://www.tutorialspoint.com/javascript/string_charat.htm) | Returns the character at the specified index. |
| [charCodeAt()](https://www.tutorialspoint.com/javascript/string_charcodeat.htm) | Returns a number indicating the Unicode value of the character at the given index. |
| [concat()](https://www.tutorialspoint.com/javascript/string_concat.htm) | Combines the text of two strings and returns a new string. |
| [indexOf()](https://www.tutorialspoint.com/javascript/string_indexof.htm) | Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found. |
| [lastIndexOf()](https://www.tutorialspoint.com/javascript/string_lastindexof.htm) | Returns the index within the calling String object of the last occurrence of the specified value, or -1 if not found. |
| [localeCompare()](https://www.tutorialspoint.com/javascript/string_localecompare.htm) | Returns a number indicating whether a reference string comes before or after or is the same as the given string in sort order. |
| [match()](https://www.tutorialspoint.com/javascript/string_match.htm) | Used to match a regular expression against a string. |
| [replace()](https://www.tutorialspoint.com/javascript/string_replace.htm) | Used to find a match between a regular expression and a string, and to replace the matched substring with a new substring. |
| [search()](https://www.tutorialspoint.com/javascript/string_search.htm) | Executes the search for a match between a regular expression and a specified string. |
| [slice()](https://www.tutorialspoint.com/javascript/string_slice.htm) | Extracts a section of a string and returns a new string. |
| [split()](https://www.tutorialspoint.com/javascript/string_split.htm) | Splits a String object into an array of strings by separating the string into substrings. |
| [substr()](https://www.tutorialspoint.com/javascript/string_substr.htm) | Returns the characters in a string beginning at the specified location through the specified number of characters. |
| [substring()](https://www.tutorialspoint.com/javascript/string_substring.htm) | Returns the characters in a string between two indexes into the string. |
| [toLocaleLowerCase()](https://www.tutorialspoint.com/javascript/string_tolocalelowercase.htm) | The characters within a string are converted to lower case while respecting the current locale. |
| [toLocaleUpperCase()](https://www.tutorialspoint.com/javascript/string_tolocaleuppercase.htm) | The characters within a string are converted to upper case while respecting the current locale. |
| [toLowerCase()](https://www.tutorialspoint.com/javascript/string_tolowercase.htm) | Returns the calling string value converted to lower case. |
| [toString()](https://www.tutorialspoint.com/javascript/string_tostring.htm) | Returns a string representing the specified object. |
| [toUpperCase()](https://www.tutorialspoint.com/javascript/string_touppercase.htm) | Returns the calling string value converted to uppercase. |
| [valueOf()](https://www.tutorialspoint.com/javascript/string_valueof.htm) | Returns the primitive value of the specified object. |

String HTML Wrappers

Here is a list of the methods that return a copy of the string wrapped inside an appropriate HTML tag.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [anchor()](https://www.tutorialspoint.com/javascript/string_anchor.htm) | Creates an HTML anchor that is used as a hypertext target. |
| [big()](https://www.tutorialspoint.com/javascript/string_big.htm) | Creates a string to be displayed in a big font as if it were in a <big> tag. |
| [blink()](https://www.tutorialspoint.com/javascript/string_blink.htm) | Creates a string to blink as if it were in a <blink> tag. |
| [bold()](https://www.tutorialspoint.com/javascript/string_bold.htm) | Creates a string to be displayed as bold as if it were in a <b> tag. |
| [fixed()](https://www.tutorialspoint.com/javascript/string_fixed.htm) | Causes a string to be displayed in fixed-pitch font as if it were in a <tt> tag |
| [fontcolor()](https://www.tutorialspoint.com/javascript/string_fontcolor.htm) | Causes a string to be displayed in the specified color as if it were in a <font color="color"> tag. |
| [fontsize()](https://www.tutorialspoint.com/javascript/string_fontsize.htm) | Causes a string to be displayed in the specified font size as if it were in a <font size="size"> tag. |
| [italics()](https://www.tutorialspoint.com/javascript/string_italics.htm) | Causes a string to be italic, as if it were in an <i> tag. |
| [link()](https://www.tutorialspoint.com/javascript/string_link.htm) | Creates an HTML hypertext link that requests another URL. |
| [small()](https://www.tutorialspoint.com/javascript/string_small.htm) | Causes a string to be displayed in a small font, as if it were in a <small> tag. |
| [strike()](https://www.tutorialspoint.com/javascript/string_strike.htm) | Causes a string to be displayed as struck-out text, as if it were in a <strike> tag. |
| [sub()](https://www.tutorialspoint.com/javascript/string_sub.htm) | Causes a string to be displayed as a subscript, as if it were in a <sub> tag |
| [sup()](https://www.tutorialspoint.com/javascript/string_sup.htm) | Causes a string to be displayed as a superscript, as if it were in a <sup> tag |

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

# JavaScript - The Arrays Object

The **Array** object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Syntax

Use the following syntax to create an **Array** object −

var fruits = new Array( "apple", "orange", "mango" );

The **Array** parameter is a list of strings or integers. When you specify a single numeric parameter with the Array constructor, you specify the initial length of the array. The maximum length allowed for an array is 4,294,967,295.

You can create array by simply assigning values as follows −

var fruits = [ "apple", "orange", "mango" ];

You will use ordinal numbers to access and to set values inside an array as follows.

fruits[0] is the first element

fruits[1] is the second element

fruits[2] is the third element

Array Properties

Here is a list of the properties of the Array object along with their description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.tutorialspoint.com/javascript/array_constructor.htm) | Returns a reference to the array function that created the object. |
| index | The property represents the zero-based index of the match in the string |
| input | This property is only present in arrays created by regular expression matches. |
| [length](https://www.tutorialspoint.com/javascript/array_length.htm) | Reflects the number of elements in an array. |
| [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm) | The prototype property allows you to add properties and methods to an object. |

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

Array Methods

Here is a list of the methods of the Array object along with their description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [concat()](https://www.tutorialspoint.com/javascript/array_concat.htm) | Returns a new array comprised of this array joined with other array(s) and/or value(s). |
| [every()](https://www.tutorialspoint.com/javascript/array_every.htm) | Returns true if every element in this array satisfies the provided testing function. |
| [filter()](https://www.tutorialspoint.com/javascript/array_filter.htm) | Creates a new array with all of the elements of this array for which the provided filtering function returns true. |
| [forEach()](https://www.tutorialspoint.com/javascript/array_foreach.htm) | Calls a function for each element in the array. |
| [indexOf()](https://www.tutorialspoint.com/javascript/array_indexof.htm) | Returns the first (least) index of an element within the array equal to the specified value, or -1 if none is found. |
| [join()](https://www.tutorialspoint.com/javascript/array_join.htm) | Joins all elements of an array into a string. |
| [lastIndexOf()](https://www.tutorialspoint.com/javascript/array_lastindexof.htm) | Returns the last (greatest) index of an element within the array equal to the specified value, or -1 if none is found. |
| [map()](https://www.tutorialspoint.com/javascript/array_map.htm) | Creates a new array with the results of calling a provided function on every element in this array. |
| [pop()](https://www.tutorialspoint.com/javascript/array_pop.htm) | Removes the last element from an array and returns that element. |
| [push()](https://www.tutorialspoint.com/javascript/array_push.htm) | Adds one or more elements to the end of an array and returns the new length of the array. |
| [reduce()](https://www.tutorialspoint.com/javascript/array_reduce.htm) | Apply a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value. |
| [reduceRight()](https://www.tutorialspoint.com/javascript/array_reduceright.htm) | Apply a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value. |
| [reverse()](https://www.tutorialspoint.com/javascript/array_reverse.htm) | Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first. |
| [shift()](https://www.tutorialspoint.com/javascript/array_shift.htm) | Removes the first element from an array and returns that element. |
| [slice()](https://www.tutorialspoint.com/javascript/array_slice.htm) | Extracts a section of an array and returns a new array. |
| [some()](https://www.tutorialspoint.com/javascript/array_some.htm) | Returns true if at least one element in this array satisfies the provided testing function. |
| [toSource()](https://www.tutorialspoint.com/javascript/array_tosource.htm) | Represents the source code of an object |
| [sort()](https://www.tutorialspoint.com/javascript/array_sort.htm) | Sorts the elements of an array |
| [splice()](https://www.tutorialspoint.com/javascript/array_splice.htm) | Adds and/or removes elements from an array. |
| [toString()](https://www.tutorialspoint.com/javascript/array_tostring.htm) | Returns a string representing the array and its elements. |
| [unshift()](https://www.tutorialspoint.com/javascript/array_unshift.htm) | Adds one or more elements to the front of an array and returns the new length of the array. |

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

# JavaScript - The Date Object

The Date object is a datatype built into the JavaScript language. Date objects are created with the **new Date( )** as shown below.

Once a Date object is created, a number of methods allow you to operate on it. Most methods simply allow you to get and set the year, month, day, hour, minute, second, and millisecond fields of the object, using either local time or UTC (universal, or GMT) time.

The ECMAScript standard requires the Date object to be able to represent any date and time, to millisecond precision, within 100 million days before or after 1/1/1970. This is a range of plus or minus 273,785 years, so JavaScript can represent date and time till the year 275755.

Syntax

You can use any of the following syntaxes to create a Date object using Date() constructor.

new Date( )

new Date(milliseconds)

new Date(datestring)

new Date(year,month,date[,hour,minute,second,millisecond ])

**Note** − Parameters in the brackets are always optional.

Here is a description of the parameters −

* **No Argument** − With no arguments, the Date() constructor creates a Date object set to the current date and time.
* **milliseconds** − When one numeric argument is passed, it is taken as the internal numeric representation of the date in milliseconds, as returned by the getTime() method. For example, passing the argument 5000 creates a date that represents five seconds past midnight on 1/1/70.
* **datestring** − When one string argument is passed, it is a string representation of a date, in the format accepted by the **Date.parse()** method.
* **7 agruments** − To use the last form of the constructor shown above. Here is a description of each argument:
  + **year** − Integer value representing the year. For compatibility (in order to avoid the Y2K problem), you should always specify the year in full; use 1998, rather than 98.
  + **month** − Integer value representing the month, beginning with 0 for January to 11 for December.
  + **date** − Integer value representing the day of the month.
  + **hour** − Integer value representing the hour of the day (24-hour scale).
  + **minute** − Integer value representing the minute segment of a time reading.
  + **second** − Integer value representing the second segment of a time reading.
  + **millisecond** − Integer value representing the millisecond segment of a time reading.

Date Properties

Here is a list of the properties of the Date object along with their description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.tutorialspoint.com/javascript/date_constructor.htm) | Specifies the function that creates an object's prototype. |
| [prototype](https://www.tutorialspoint.com/javascript/object_prototype.htm) | The prototype property allows you to add properties and methods to an object |

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

Date Methods

Here is a list of the methods used with **Date** and their description.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [Date()](https://www.tutorialspoint.com/javascript/date_date.htm) | Returns today's date and time |
| [getDate()](https://www.tutorialspoint.com/javascript/date_getdate.htm) | Returns the day of the month for the specified date according to local time. |
| [getDay()](https://www.tutorialspoint.com/javascript/date_getday.htm) | Returns the day of the week for the specified date according to local time. |
| [getFullYear()](https://www.tutorialspoint.com/javascript/date_getfullyear.htm) | Returns the year of the specified date according to local time. |
| [getHours()](https://www.tutorialspoint.com/javascript/date_gethours.htm) | Returns the hour in the specified date according to local time. |
| [getMilliseconds()](https://www.tutorialspoint.com/javascript/date_getmilliseconds.htm) | Returns the milliseconds in the specified date according to local time. |
| [getMinutes()](https://www.tutorialspoint.com/javascript/date_getminutes.htm) | Returns the minutes in the specified date according to local time. |
| [getMonth()](https://www.tutorialspoint.com/javascript/date_getmonth.htm) | Returns the month in the specified date according to local time. |
| [getSeconds()](https://www.tutorialspoint.com/javascript/date_getseconds.htm) | Returns the seconds in the specified date according to local time. |
| [getTime()](https://www.tutorialspoint.com/javascript/date_gettime.htm) | Returns the numeric value of the specified date as the number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| [getTimezoneOffset()](https://www.tutorialspoint.com/javascript/date_gettimezoneoffset.htm) | Returns the time-zone offset in minutes for the current locale. |
| [getUTCDate()](https://www.tutorialspoint.com/javascript/date_getutcdate.htm) | Returns the day (date) of the month in the specified date according to universal time. |
| [getUTCDay()](https://www.tutorialspoint.com/javascript/date_getutcday.htm) | Returns the day of the week in the specified date according to universal time. |
| [getUTCFullYear()](https://www.tutorialspoint.com/javascript/date_getutcfullyear.htm) | Returns the year in the specified date according to universal time. |
| [getUTCHours()](https://www.tutorialspoint.com/javascript/date_getutchours.htm) | Returns the hours in the specified date according to universal time. |
| [getUTCMilliseconds()](https://www.tutorialspoint.com/javascript/date_getutcmilliseconds.htm) | Returns the milliseconds in the specified date according to universal time. |
| [getUTCMinutes()](https://www.tutorialspoint.com/javascript/date_getutcminutes.htm) | Returns the minutes in the specified date according to universal time. |
| [getUTCMonth()](https://www.tutorialspoint.com/javascript/date_getutcmonth.htm) | Returns the month in the specified date according to universal time. |
| [getUTCSeconds()](https://www.tutorialspoint.com/javascript/date_getutcseconds.htm) | Returns the seconds in the specified date according to universal time. |
| [getYear()](https://www.tutorialspoint.com/javascript/date_getyear.htm) | **Deprecated** - Returns the year in the specified date according to local time. Use getFullYear instead. |
| [setDate()](https://www.tutorialspoint.com/javascript/date_setdate.htm) | Sets the day of the month for a specified date according to local time. |
| [setFullYear()](https://www.tutorialspoint.com/javascript/date_setfullyear.htm) | Sets the full year for a specified date according to local time. |
| [setHours()](https://www.tutorialspoint.com/javascript/date_sethours.htm) | Sets the hours for a specified date according to local time. |
| [setMilliseconds()](https://www.tutorialspoint.com/javascript/date_setmilliseconds.htm) | Sets the milliseconds for a specified date according to local time. |
| [setMinutes()](https://www.tutorialspoint.com/javascript/date_setminutes.htm) | Sets the minutes for a specified date according to local time. |
| [setMonth()](https://www.tutorialspoint.com/javascript/date_setmonth.htm) | Sets the month for a specified date according to local time. |
| [setSeconds()](https://www.tutorialspoint.com/javascript/date_setseconds.htm) | Sets the seconds for a specified date according to local time. |
| [setTime()](https://www.tutorialspoint.com/javascript/date_settime.htm) | Sets the Date object to the time represented by a number of milliseconds since January 1, 1970, 00:00:00 UTC. |
| [setUTCDate()](https://www.tutorialspoint.com/javascript/date_setutcdate.htm) | Sets the day of the month for a specified date according to universal time. |
| [setUTCFullYear()](https://www.tutorialspoint.com/javascript/date_setutcfullyear.htm) | Sets the full year for a specified date according to universal time. |
| [setUTCHours()](https://www.tutorialspoint.com/javascript/date_setutchours.htm) | Sets the hour for a specified date according to universal time. |
| [setUTCMilliseconds()](https://www.tutorialspoint.com/javascript/date_setutcmilliseconds.htm) | Sets the milliseconds for a specified date according to universal time. |
| [setUTCMinutes()](https://www.tutorialspoint.com/javascript/date_setutcminutes.htm) | Sets the minutes for a specified date according to universal time. |
| [setUTCMonth()](https://www.tutorialspoint.com/javascript/date_setutcmonth.htm) | Sets the month for a specified date according to universal time. |
| [setUTCSeconds()](https://www.tutorialspoint.com/javascript/date_setutcseconds.htm) | Sets the seconds for a specified date according to universal time. |
| [setYear()](https://www.tutorialspoint.com/javascript/date_setyear.htm) | **Deprecated -** Sets the year for a specified date according to local time. Use setFullYear instead. |
| [toDateString()](https://www.tutorialspoint.com/javascript/date_todatestring.htm) | Returns the "date" portion of the Date as a human-readable string. |
| [toGMTString()](https://www.tutorialspoint.com/javascript/date_togmtstring.htm) | **Deprecated -** Converts a date to a string, using the Internet GMT conventions. Use toUTCString instead. |
| [toLocaleDateString()](https://www.tutorialspoint.com/javascript/date_tolocaledatestring.htm) | Returns the "date" portion of the Date as a string, using the current locale's conventions. |
| [toLocaleFormat()](https://www.tutorialspoint.com/javascript/date_tolocaleformat.htm) | Converts a date to a string, using a format string. |
| [toLocaleString()](https://www.tutorialspoint.com/javascript/date_tolocalestring.htm) | Converts a date to a string, using the current locale's conventions. |
| [toLocaleTimeString()](https://www.tutorialspoint.com/javascript/date_tolocaletimestring.htm) | Returns the "time" portion of the Date as a string, using the current locale's conventions. |
| [toSource()](https://www.tutorialspoint.com/javascript/date_tosource.htm) | Returns a string representing the source for an equivalent Date object; you can use this value to create a new object. |
| [toString()](https://www.tutorialspoint.com/javascript/date_tostring.htm) | Returns a string representing the specified Date object. |
| [toTimeString()](https://www.tutorialspoint.com/javascript/date_totimestring.htm) | Returns the "time" portion of the Date as a human-readable string. |
| [toUTCString()](https://www.tutorialspoint.com/javascript/date_toutcstring.htm) | Converts a date to a string, using the universal time convention. |
| [valueOf()](https://www.tutorialspoint.com/javascript/date_valueof.htm) | Returns the primitive value of a Date object. |

Converts a date to a string, using the universal time convention.

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( )](https://www.tutorialspoint.com/javascript/date_parse.htm) | Parses a string representation of a date and time and returns the internal millisecond representation of that date. |
| [Date.UTC( )](https://www.tutorialspoint.com/javascript/date_utc.htm) | 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.

# JavaScript - The Math Object

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 \](https://www.tutorialspoint.com/javascript/math_e.htm) | Euler's constant and the base of natural logarithms, approximately 2.718. |
| [LN2](https://www.tutorialspoint.com/javascript/math_ln2.htm) | Natural logarithm of 2, approximately 0.693. |
| [LN10](https://www.tutorialspoint.com/javascript/math_ln10.htm) | Natural logarithm of 10, approximately 2.302. |
| [LOG2E](https://www.tutorialspoint.com/javascript/math_log2e.htm) | Base 2 logarithm of E, approximately 1.442. |
| [LOG10E](https://www.tutorialspoint.com/javascript/math_log10e.htm) | Base 10 logarithm of E, approximately 0.434. |
| [PI](https://www.tutorialspoint.com/javascript/math_pi.htm) | Ratio of the circumference of a circle to its diameter, approximately 3.14159. |
| [SQRT1\_2](https://www.tutorialspoint.com/javascript/math_sqrt1_2.htm) | Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707. |
| [SQRT2](https://www.tutorialspoint.com/javascript/math_sqrt2.htm) | 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 Methods

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| [abs()](https://www.tutorialspoint.com/javascript/math_abs.htm) | Returns the absolute value of a number. |
| [acos()](https://www.tutorialspoint.com/javascript/math_acos.htm) | Returns the arccosine (in radians) of a number. |
| [asin()](https://www.tutorialspoint.com/javascript/math_asin.htm) | Returns the arcsine (in radians) of a number. |
| [atan()](https://www.tutorialspoint.com/javascript/math_atan.htm) | Returns the arctangent (in radians) of a number. |
| [atan2()](https://www.tutorialspoint.com/javascript/math_atan2.htm) | Returns the arctangent of the quotient of its arguments. |
| [ceil()](https://www.tutorialspoint.com/javascript/math_ceil.htm) | Returns the smallest integer greater than or equal to a number. |
| [cos()](https://www.tutorialspoint.com/javascript/math_cos.htm) | Returns the cosine of a number. |
| [exp()](https://www.tutorialspoint.com/javascript/math_exp.htm) | Returns EN, where N is the argument, and E is Euler's constant, the base of the natural logarithm. |
| [floor()](https://www.tutorialspoint.com/javascript/math_floor.htm) | Returns the largest integer less than or equal to a number. |
| [log()](https://www.tutorialspoint.com/javascript/math_log.htm) | Returns the natural logarithm (base E) of a number. |
| [max()](https://www.tutorialspoint.com/javascript/math_max.htm) | Returns the largest of zero or more numbers. |
| [min()](https://www.tutorialspoint.com/javascript/math_min.htm) | Returns the smallest of zero or more numbers. |
| [pow()](https://www.tutorialspoint.com/javascript/math_pow.htm) | Returns base to the exponent power, that is, base exponent. |
| [random()](https://www.tutorialspoint.com/javascript/math_random.htm) | Returns a pseudo-random number between 0 and 1. |
| [round()](https://www.tutorialspoint.com/javascript/math_round.htm) | Returns the value of a number rounded to the nearest integer. |
| [sin()](https://www.tutorialspoint.com/javascript/math_sin.htm) | Returns the sine of a number. |
| [sqrt()](https://www.tutorialspoint.com/javascript/math_sqrt.htm) | Returns the square root of a number. |
| [tan()](https://www.tutorialspoint.com/javascript/math_tan.htm) | Returns the tangent of a number. |
| [toSource()](https://www.tutorialspoint.com/javascript/math_tosource.htm) | Returns the string "Math". |

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

# Regular Expressions and RegExp Object

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** |
| [...] | Any one character between the brackets. |
| [^...] | Any one character not between the brackets. |
| [0-9] | 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 one or more p's. |
| p\* | It matches any string containing zero or more p's. |
| p? | It matches any string containing at most one p. |
| 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. |
| <b>(.\*)</b> | It matches any string enclosed within <b> and </b>. |
| 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 |
| \cX | The control character ^X; for example, \cJ is equivalent to the newline character \n |

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** |
| . | a single character |
| \s | a whitespace character (space, tab, newline) |
| \S | non-whitespace character |
| \d | a digit (0-9) |
| \D | a non-digit |
| \w | a word character (a-z, A-Z, 0-9, \_) |
| \W | a non-word character |
| [\b] | a literal backspace (special case). |
| [aeiou] | matches a single character in the given set |
| [^aeiou] | matches a single character outside the given set |
| (foo|bar|baz) | 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** |
| I | Perform 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 matchthat 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](https://www.tutorialspoint.com/javascript/regexp_constructor.htm) | Specifies the function that creates an object's prototype. |
| [global](https://www.tutorialspoint.com/javascript/regexp_global.htm) | Specifies if the "g" modifier is set. |
| [ignoreCase](https://www.tutorialspoint.com/javascript/regexp_ignorecase.htm) | Specifies if the "i" modifier is set. |
| [lastIndex](https://www.tutorialspoint.com/javascript/regexp_lastindex.htm) | The index at which to start the next match. |
| [multiline](https://www.tutorialspoint.com/javascript/regexp_multiline.htm) | Specifies if the "m" modifier is set. |
| [source](https://www.tutorialspoint.com/javascript/regexp_source.htm) | The text of the pattern. |

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

RegExp Methods

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

|  |  |
| --- | --- |
| **Method** | **Description** |
| [exec()](https://www.tutorialspoint.com/javascript/regexp_exec.htm) | Executes a search for a match in its string parameter. |
| [test()](https://www.tutorialspoint.com/javascript/regexp_test.htm) | Tests for a match in its string parameter. |
| [toSource()](https://www.tutorialspoint.com/javascript/regexp_tosource.htm) | Returns an object literal representing the specified object; you can use this value to create a new object. |
| [toString()](https://www.tutorialspoint.com/javascript/regexp_tostring.htm) | Returns a string representing the specified object. |

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

# JavaScript - Document Object Model or 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](https://www.tutorialspoint.com/javascript/javascript_legacy_dom.htm) − 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](https://www.tutorialspoint.com/javascript/javascript_w3c_dom.htm) − 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](https://www.tutorialspoint.com/javascript/javascript_ie4_dom.htm) − 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.

DOM compatibility

If you want to write a script with the flexibility to use either W3C DOM or IE 4 DOM depending on their availability, then you can use a capability-testing approach that first checks for the existence of a method or property to determine whether the browser has the capability you desire. For example −

if (document.getElementById) {

// If the W3C method exists, use it

}

else if (document.all) {

// If the all[] array exists, use it

}

else {

// Otherwise use the legacy DOM

}