 Javascript is the most popular **programming language** in the world and that makes it a programmer’s great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end softwares using different Javascript based frameworks like jQuery, Node.JS etc.

 Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox , Safari and every browser you know as of today, supports Javascript.

 Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.

 JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.

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.

**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 multi-threading 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.

## 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.

[Live Demo](http://tpcg.io/wAb0EM)

<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 Operators**

JavaScript supports the following types of operators.

* Arithmetic Operators
* Comparison Operators
* Logical (or Relational) Operators
* Assignment Operators
* Conditional (or ternary) Operators
* **Arithmetic Operators**
* JavaScript supports the following arithmetic operators −
* Assume variable A holds 10 and variable B holds 20, then −

|  |  |
| --- | --- |
| **Sr.No.** | **Operator & 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".

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 & 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 & 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 & 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 & 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

**Clint side Validation**

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

* **Basic Validation** − First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
* **Data Format Validation** − Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

### Example

We will take an example to understand the process of validation. Here is a simple form in html format.

<html>

<head>

<title>Form Validation</title>

<script type = "text/javascript">

<!--

// Form validation code will come here.

//-->

</script>

</head>

<body>

<form action = "/cgi-bin/test.cgi" name = "myForm" onsubmit = "return(validate());">

<table cellspacing = "2" cellpadding = "2" border = "1">

<tr>

<td align = "right">Name</td>

<td><input type = "text" name = "Name" /></td>

</tr>

<tr>

<td align = "right">EMail</td>

<td><input type = "text" name = "EMail" /></td>

</tr>

<tr>

<td align = "right">Zip Code</td>

<td><input type = "text" name = "Zip" /></td>

</tr>

<tr>

<td align = "right">Country</td>

<td>

<select name = "Country">

<option value = "-1" selected>[choose yours]</option>

<option value = "1">USA</option>

<option value = "2">UK</option>

<option value = "3">INDIA</option>

</select>

</td>

</tr>

<tr>

<td align = "right"></td>

<td><input type = "submit" value = "Submit" /></td>

</tr>

</table>

</form>

</body>

</html>

### Output

## Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

<script type = "text/javascript">

<!--

// Form validation code will come here.

function validate() {

if( document.myForm.Name.value == "" ) {

alert( "Please provide your name!" );

document.myForm.Name.focus() ;

return false;

}

if( document.myForm.EMail.value == "" ) {

alert( "Please provide your Email!" );

document.myForm.EMail.focus() ;

return false;

}

if( document.myForm.Zip.value == "" || isNaN( document.myForm.Zip.value ) ||

document.myForm.Zip.value.length != 5 ) {

alert( "Please provide a zip in the format #####." );

document.myForm.Zip.focus() ;

return false;

}

if( document.myForm.Country.value == "-1" ) {

alert( "Please provide your country!" );

return false;

}

return( true );

}

//-->

</script>

## Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a ‘@’ sign and a dot (.). Also, the ‘@’ must not be the first character of the email address, and the last dot must at least be one character after the ‘@’ sign.

### Example

Try the following code for email validation.

<script type = "text/javascript">

<!--

function validateEmail() {

var emailID = document.myForm.EMail.value;

atpos = emailID.indexOf("@");

dotpos = emailID.lastIndexOf(".");

if (atpos < 1 || ( dotpos - atpos < 2 )) {

alert("Please enter correct email ID")

document.myForm.EMail.focus() ;

return false;

}

return( true );

}

//-->

</script>

**Function**

* Function is a reusable code-block that will be executed whenever it is called.
* Function is a great time saver.
* It is used for performing repetitive tasks where you can call the same function multiple times to get the same effect.
* It allows code reusability.
* JavaScript provides number of built-in functions.

## Common Built-in Functions

|  |  |
| --- | --- |
| **Functions** | **Description** |
| isNan() | **Returns true,** if the object is Not a Number. **Returns false,** if the object is a number. |
| parseFloat (string) | If the string begins with a number, the function reads through the string until it finds the end of the number; cuts off the remainder of the string and returns the result. If the string does not begin with a number, the function returns NaN. |
| parseInt (string) | If the string begins with an integer, the function reads through the string until it finds the end of the integer, cuts off the remainder of the string and returns the result. If the string does not begin with an integer, the function returns NaN (Not a Number). |
| String (object) | Converts the object into a string. |
| eval() | Returns the result of evaluating an arithmetic expression. |

## User-defined Functions

* User-defined function means you can create a function for your own use. You can create yourself according to your need.
* In JavaScript, these functions are written in between the <HEAD> tag of the HTML page.

**Syntax:**  
function function\_name()  
{  
     //Code;  
}

#### Example : Function Declaration

function add()      
{  
     var a, b;  
     var sum = 0;  
     sum = a + b;  
     document.write(“Addition : ”+sum);  
}

## How is a function executed on an event in JavaScript?

<input type= “button” onClick= “add()” value= “buttonvalue”>

* onClick – Event Handler
* add() – Function name

#### Example : Simple Program on User-defined Function

<html>   
<body>   
     <script type="text/javascript">   
     function add()          // Function Declaration  
     {   
          var a = 2,b = 3;   
          var sum = 0;   
          sum = a+b;   
          document.write("<b>Addition: </b>"+sum);   
     }   
     </script>   
     <p> Click the Button</p>   
     <input type="button" onClick="add()" value="Click">          //add() - Calling Function  
</body>   
</html>

**Output:**  
  
**Addition:** 5

v

Javascript also has its own built in functions. These built in functions also known as global functions.

The beauty of these function are, you can use these function with any Javascript built in object such as String , Number, Date, RegExp, Array, Boolean and Math. Javascript's all built in object support these functions.

## Javascript Built in Functions

Here is a list of built in functions with description.

|  |  |
| --- | --- |
| **Function** | **Description** |
| [isNaN](https://www.techstrikers.com/Javascript/javascript-isnan-function.php) | This function is intended to determines whether value is a legal number or not. |
| [isFinite](https://www.techstrikers.com/Javascript/javascript-isfinite-function.php) | This function is intended to find whether a number is a finite legal number. |
| [eval](https://www.techstrikers.com/Javascript/javascript-eval-function.php) | This function is intended to execute Javascript source code. |
| [number](https://www.techstrikers.com/Javascript/javascript-number-function.php) | This function is intended to converts object to the corresponding number value. |
| [string](https://www.techstrikers.com/Javascript/javascript-string-function.php) | This function is intended to converts object to the corresponding string value. |
| [parseInt](https://www.techstrikers.com/Javascript/javascript-parseint-function.php) | This function is intended to converts string value to integer. |
| [parseFloat](https://www.techstrikers.com/Javascript/javascript-parsefloat-function.php) | This function is intended to converts string value to floating point number. |
| [escape](https://www.techstrikers.com/Javascript/javascript-escape-function.php) | This function is intended to encodes the string value into world wide acceptable format. |
| [encodeURI](https://www.techstrikers.com/Javascript/javascript-encodeuri-function.php) | This function is intended to encode URI. |
| [decodeURI](https://www.techstrikers.com/Javascript/javascript-decodeuri-function.php) | This function is intended to decode URI. |
| [encodeURIComponent](https://www.techstrikers.com/Javascript/javascript-encodeuricomponent-function.php) | This function is intended to encode URI component. |
| [decodeURIComponent](https://www.techstrikers.com/Javascript/javascript-decodeuricomponent-function.php) | This function is intended to decode URI component. |

**Javascript Built-in-funtion Object**

In Javascript string object is used to hold a series of charecters and allows you to perform string manipulations, such as finding character index, replacing charecter, converting string in uppercase ot lowercase etc.

A Javascript can be any text inside double or single quotes and support variety of helper methods to manipulate on it.

**Syntax**

<html>

<head>

<title>My first Javascript code</title>

var stringOne = "Javascript suppert variety of helper methods";

var stringTwo = "Javascript is case sensitive language";

</head>

<body>

</body>

</html>

OR, you can use String global object directly

<html>

<head>

<title>My first Javascript code</title>

var obj = new String("Javascript suppert variety of helper methods");

</head>

<body>

</body>

</html>

**Distinction between string primitives and String objects**

JavaScript distinguishes between String objects and primitive string values. JavaScript automatically converts primitives value to String objects, so that it's allows you to use String object methods for primitive strings as well.

<html>

<head>

<title>My first Javascript code</title>

var string\_prim = 'Some string';

var string\_obj = new String(string\_prim);

alert(typeof string\_prim); // display "string"

alert(typeof string\_obj); // display "object"

</head>

<body>

</body>

</html>

In Javascript string primitives and string objects also give different results when you use eval(). Primitives passed to eval are treated as source code; String objects are treated as all other objects are, by returning the object. For example:

<html>

<head>

<title>My first Javascript code</title>

var str1 = '12 + 12'; // creates a string primitive

var str2 = new String('23 + 23'); // creates a String object

alert(eval(s1)); // returns the number 24

alert(eval(s2)); // returns the string "23 + 23"

</head>

<body>

</body>

</html>

# Javascript Number Object

In Javascript number type include floating-point and integer values, its wrapper for primitive numeric values.

A Number object is created using the Number() constructor. In Javascript numbers can be divided into two groups:

**Integers** - The numbers such as 120, -232, and 5.

**floating-point** - The fractional numbers such as 4.33, -8.422, and .55

In Javascript primitive number can be declared using the literal notation:

**Syntax**

<html>

<head>

<title>My first Javascript code</title>

var x = 13.44; // A number with decimals

var y = 45; // A number without decimals

</head>

<body>

</body>

</html>

OR, you can use Number global object directly

<html>

<head>

<title>My first Javascript code</title>

var val = new Number(number);

</head>

<body>

</body>

</html>

## Number Properties

Number object has following properties that are part of every object.

|  |  |
| --- | --- |
| **Property** | **Description** |
| MAX\_VALUE | Return largest positive representable number in Javascript. |
| MIN\_VALUE | The smallest positive representable number in Javascript. |
| NEGATIVE\_INFINITY | Represents negative infinity (returned on overflow) |
| NaN | Represents a "Not-a-Number" value |
| POSITIVE\_INFINITY | Represents positive infinity (returned on overflow) |
| prototype | Allows to add addition of properties and function to all objects |
| constructor | Returns a reference to the function that created the Number object |

# Javascript Math Object

In Javascript Math object provides veriety of methods or function and properies that can be used to easily do mathematical calculations. Math object is an in-built object and so it is not required to instantiate the object as done for Date object.

Unlike the other global objects, Math is not a constructor. All properties and methods of Math are static.

Here is the simple syntax to call properties and methods of Math.

**Syntax**

<html>

<head>

<title>Javascript Math Object</title>

var log\_val = Math.LOG10E;

var sine\_val = Math.sin(10);

</head>

<body>

</body>

</html>

## Math Properties

Here is a list of each math property with description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [E](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Euler's constant and the base of natural logarithms, approximately 2.718. |
| [LN2](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Natural logarithm of 2, approximately 0.693. |
| [LN10](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Natural logarithm of 10, approximately 2.302. |
| [LOG2E](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Base 2 logarithm of E, approximately 1.442. |
| [LOG10E](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Base 10 logarithm of E, approximately 0.434. |
| [PI](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Ratio of the circumference of a circle to its diameter, approximately 3.14159. |
| [SQRT1\_2](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Square root of 1/2; equivalently, 1 over the square root of 2, approximately 0.707. |
| [SQRT2](https://www.techstrikers.com/Javascript/javascript-math-object.php) | Square root of 2, approximately 1.414. |

# Javascript Date Object

The Javascript date() method of date object returns today's date and time. Date object are usually created using new Date() constructor. Use it for example to get today's date and time.

In Javascript to work with date and time, create an instance of date() object with new keyword.

There are different way to create a date object in Javascript.

* new Date("Month dd, yyyy hh:mm:ss")
* new Date("Month dd, yyyy")
* new Date(yy,mm,dd,hh,mm,ss)
* new Date(yy,mm,dd)
* new Date(milliseconds)

Here is how you can create a Date object for each of the ways above:

**Syntax**

<html>

<head>

<title>Javascript Date Object</title>

var my\_date=new Date("August 15, 1947 13:14:00");

var my\_date=new Date("August 15, 1947");

var my\_date=new Date(88,09,12,13,14,00);

var my\_date=new Date(88,09,12);

var my\_date=new Date(500);

</head>

<body>

</body>

</html>

**Note:** Paramters in the brackets are always optional

* If no arguments are provided, the constructor creates a JavaScript Date object for the current date and time according to system settings.
* If at least two arguments are supplied, missing arguments are either set to 1 or 0 for all others.
* JavaScript date is based on a time value that is milliseconds since midnight 01 January, 1970 UTC. The JavaScript Date object range is -100,000,000 days to 100,000,000 days relative to 01 January, 1970 UTC.
* JavaScript Date object provides uniform behavior across platforms. The time value can be passed between systems to represent the same moment in time and if used to create a local date object, will reflect the local equivalent of the time.
* JavaScript Date object supports a number of UTC (universal) methods, as well as local time methods. UTC, also known as Greenwich Mean Time (GMT), refers to the time as set by the World Time Standard. The local time is the time known to the computer where JavaScript is executed.
* Invoking JavaScript Date as a function (i.e., without the new operator) will return a string representing the current date and time.

Once created an instance of the Date object, you can access all the methods of the date object from intance variable.

## Date Properties

Here is a list of each date property with description.

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.techstrikers.com/Javascript/javascript-date-object.php) | Specifies the function that creates an object's prototype. |
| [prototype](https://www.techstrikers.com/Javascript/javascript-date-object.php) | This will allows you to add properties and function to an object. |

# Javascript Array Object

In Javascript Array Object is special type of object, it is used to store multiple values in a single variable.

The elements in a JavaScript array can have different data types, and they can be accessed by indexing. Javascript allows to array of array, elements of an array can also be arrays. In that case, the array is a multi-dimensional array. JavaScript allows associative arrays, too. In this type of arrays, each stored value is associated with a unique key, which can be used as an index. The unique keys are string types.

Here is the simple syntax to call properties and functions of Array Object.

**Syntax**

<html>

<head>

<title>Javascript Array Object</title>

<SCRIPT LANGUAGE="JavaScript">

var arr = new Array ();

var arr = new Array ("Orange", "Mango", "Banana");

// or

var arr = ["Orange", "Mango", "Banana"];

// display the elements of the array

for (var i = 0; i < arr.length; i++) {

document.write (arr[i]);

}

// you can access the elements of the array by zero-based indices

var firstItem = arr[0]; // "Orange"

var thirdItem = arr[2]; // "Mango"

// you can get the length of the array by the length property

var len = arr.length; // 3

//Creating multidimensional array

var arr1 = new Array ("A", "B", "C");

var arr2 = new Array (1, 2, 3);

var multiArr = new Array (arr1, arr2);

// or

var multiArr = [arr1, arr2];

// you can access the elements of the array by zero-based indices

var firstRow = multiArr[0]; // same as arr1

var secondRowFirstCell = multiArr[1][0]; // 1

</SCRIPT>

</head>

<body>

</body>

</html>

## Array Properties

Here is a list of each array property with description.

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

# Javascript Boolean Object

Javascript boolean object can have logical values either true or false. JavaScript implicitly uses the Boolean object whenever a Boolean data type is converted to a Boolean object.

Variable type Boolean does not support any quotes for example - "true" is not equal to true, "true" will be considered as a string.

**Syntax**

<html>

<html>

<head>

<title>Javascript Boolean Object</title>

<SCRIPT LANGUAGE="JavaScript">

var val = new Boolean(value);

</SCRIPT>

</head>

<body>

</body>

</html>

JavaScript pitfalls: null, false, undefined and NaN

## Comparison operators

**Zero - 0 :** Evaluates to false in boolean operations. Always use === when comparing to the number 0.

### Javascript Boolean Example

<html>

<head>

<title>Javascript Boolean Object</title>

<SCRIPT LANGUAGE="JavaScript">

var x = 0;

var y = false;

alert(x == y) // return true

alert(x === y) // return false

</SCRIPT>

</head>

<body>

</body>

</html>

**Empty - "" :** Evaluates to false in boolean operations. Always use === when comparing to an empty string.

### Javascript Boolean Example

<html>

<head>

<title>Javascript Boolean Object</title>

<SCRIPT LANGUAGE="JavaScript">

var x = null;

var y = false;

alert(x == y) //return true

alert(x === y) //return false

</SCRIPT>

</head>

<body>

</body>

</html>

**Undefined :** If a variable has not declared or assigned yet then that variable will be given a special undefined value.

### Javascript Boolean Example

<html>

<head>

<title>Javascript Boolean Object</title>

<SCRIPT LANGUAGE="JavaScript">

var x;

var y = false;

typeof(x) //return undefined

alert(x == y) //return true

alert(x === y) //return false

</SCRIPT>

</head>

<body>

</body>

</html>

**NaN :** Not a Number, generated when arithmetic operations return invalid results.

### Javascript Boolean Example

<html>

<head>

<title>Javascript Boolean Object</title>

<SCRIPT LANGUAGE="JavaScript">

var x = 10/zero;

alert(x); //return NaN

var x = 10/zero;

alert(x == x) //return false

</SCRIPT>

</head>

<body>

</body>

</html>

## Boolean Properties

|  |  |
| --- | --- |
| **Property** | **Description** |
| [constructor](https://www.techstrikers.com/Javascript/javascript-boolean-object.php) | This will returns a reference to the Boolean function that created the object. |
| [prototype](https://www.techstrikers.com/Javascript/javascript-boolean-object.php) | This prototype property allows you to add properties and methods to an object. |

# Javascript JSON Object

JavaScript Object Notation - JSON is a very lightweight data-interchange format. The most improtant thing about JSON is, easy to read and write. It is language-independent data exchange format that makes you easy to easily share information between veriety of our systems and devices, regardless of technology.

JSON is a simple text (key-value) that is completely language independent but uses conventions that are familiar to programmers of C#, Java, JavaScript and many others. These make JSON an ideal data-interchange language. Here is an example of JSON data:

### Javascript Simple JSON Object Example

<html>

<head>

<title>My first JSON Object Code</title>

<SCRIPT LANGUAGE="JavaScript">

var jsonData=

{

"firstName": "Jimi",

"lastName": "Scotts",

"address": "12-13-283/A1"

}

document.write(jsonData.firstName + ' ' + jsonData.lastame);

</SCRIPT>

</HEAD>

<BODY>

</BODY>

</HTML>

### Javascript Array In JSON Object Example

<html>

<head>

<title>My first JSON Object Code</title>

<SCRIPT LANGUAGE="JavaScript">

var jsonData={"customer":[

{

"firstName": "Jimi",

"lastName": "Scotts",

"address": "12-13-283/A1"

},

{

"firstName": "Tom",

"lastName": "Scotts",

"address": "12-13-284/A2"

}

]}

document.write(jsonData.customer[0].firstName + ' ' + jsonData.customer[0].lastame);

</SCRIPT>

</HEAD>

<BODY>

</BODY>

</HTML>