**JAVASCRIPT**

1. **What is Javascript?**

JavaScript is a scripting or programming language that allows you to implement complex features on web pages — every time a web page does more than just sit there and display static information for you to look at — displaying timely content updates, interactive maps, animated 2D/3D graphics, scrolling video jukeboxes, etc. — you can bet that JavaScript is probably involved.

## **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.

1. **SYNTAX**

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

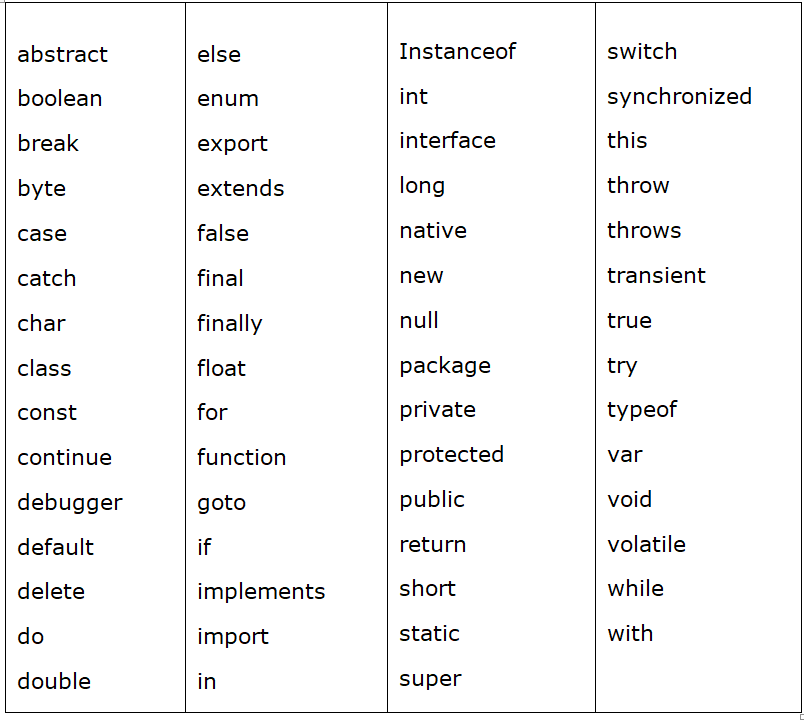
You can place the <script> tags, containing your JavaScript, anywhere within you web page, but it is normally recommended that you should keep it within the <head> tags.

<script ...>

JavaScript code

</script>

|  | **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 //-->. |
| --- | --- |
|  | **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.    **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 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 |



**JAVASCRIPT OPERATORS**

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

· Arithmetic Operators

· Comparison Operators

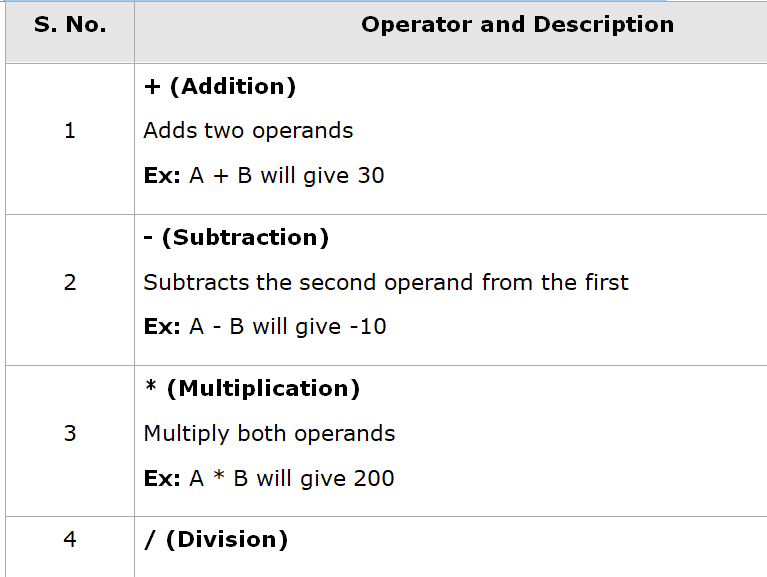
· Logical (or Relational) Operators

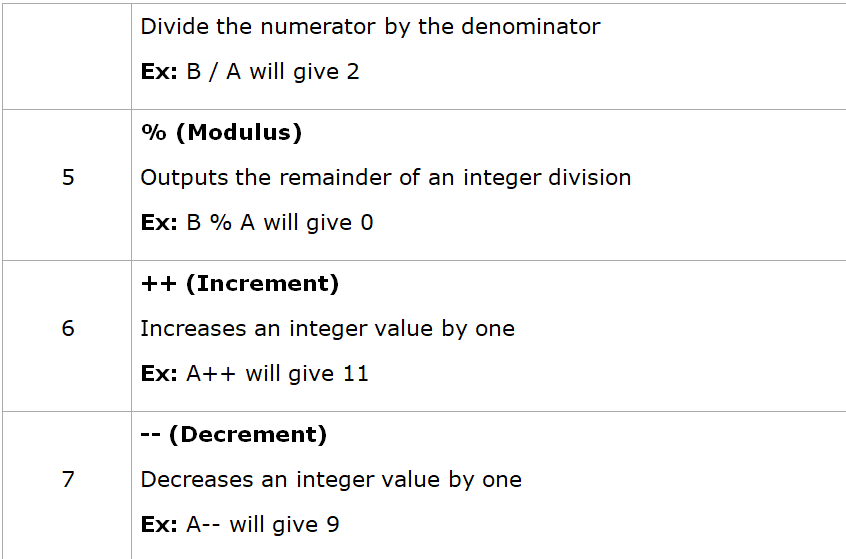
· Assignment Operators

· Conditional (or ternary) Operators

**Arithmetic Operators**

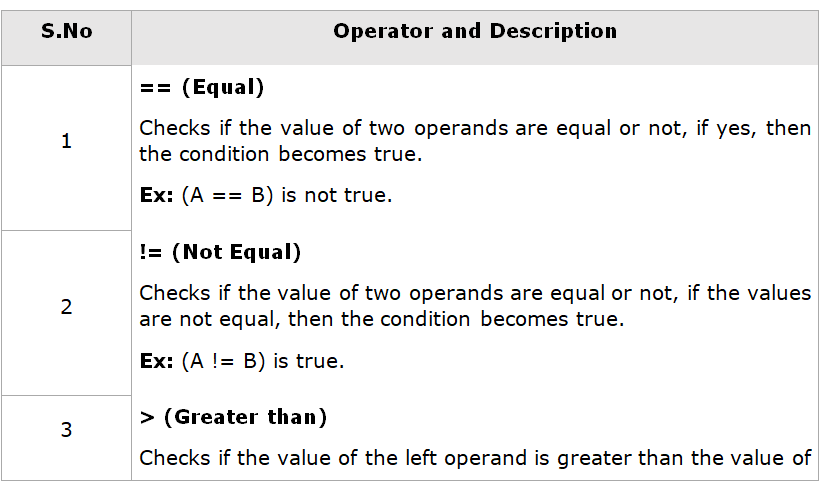
JavaScript supports the following arithmetic operators: Assume variable A holds 10 and variable B holds 20, then:

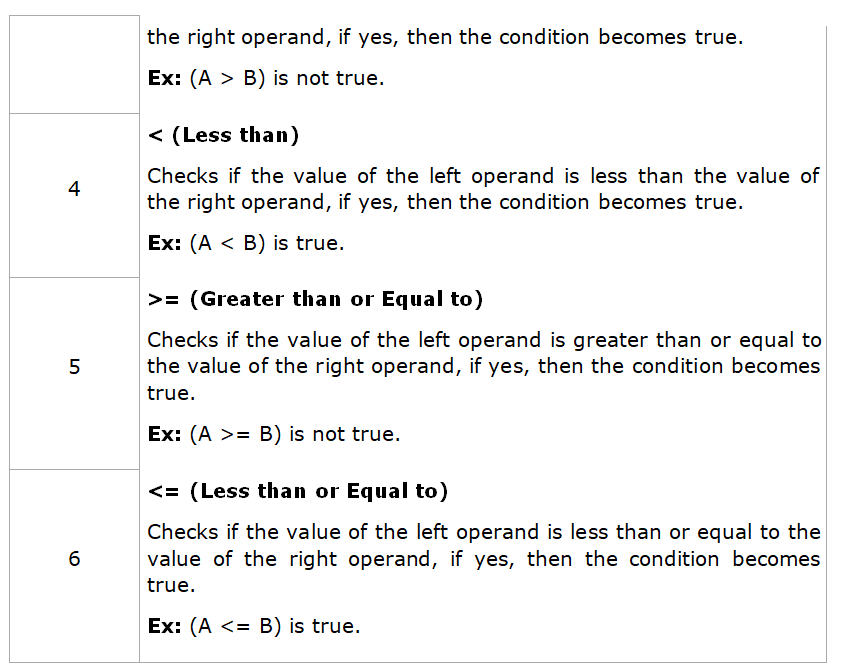




**COMPARISON OPERATORS**

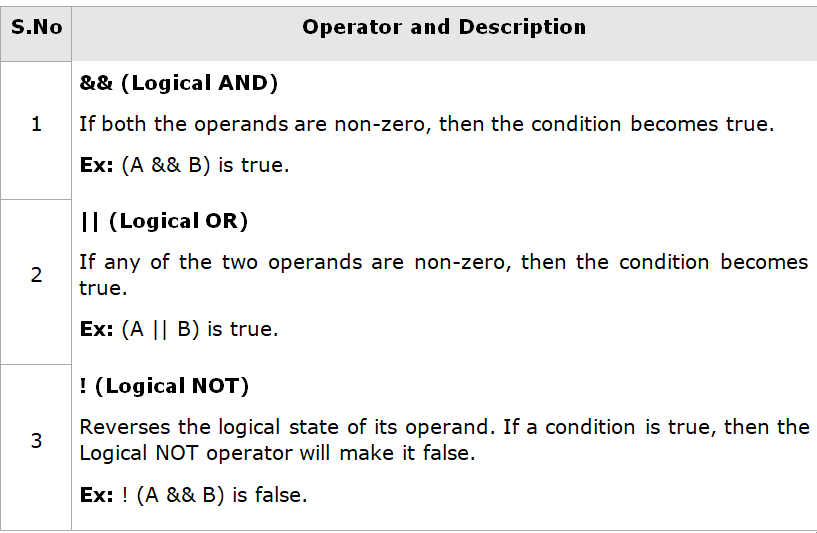
JavaScript supports the following comparison operators: Assume variable A holds 10 and variable B holds 20, then:





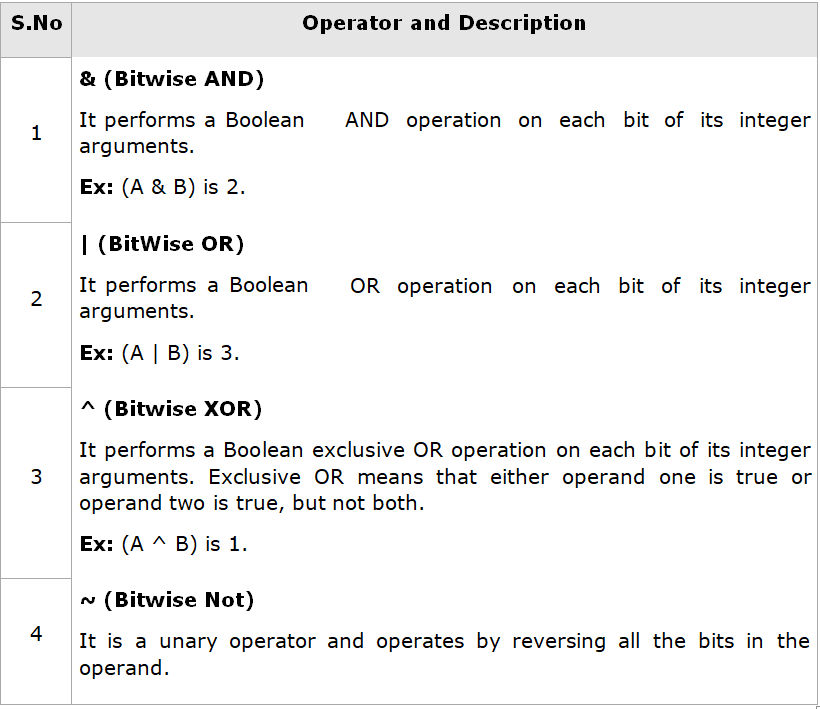
**LOGICAL OPERATORS**

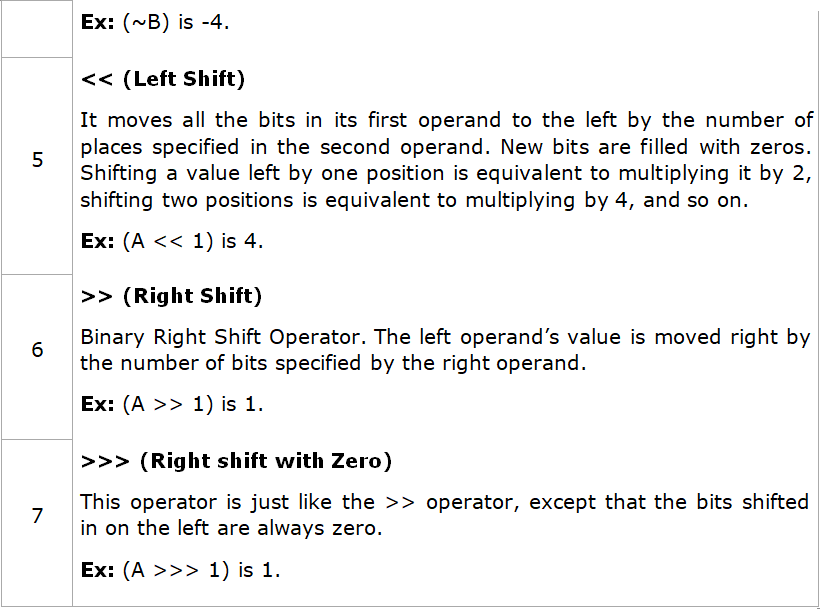
JavaScript supports the following logical operators: Assume variable A holds 10 and variable B holds 20, then:



**BITWISE OPERATORS**

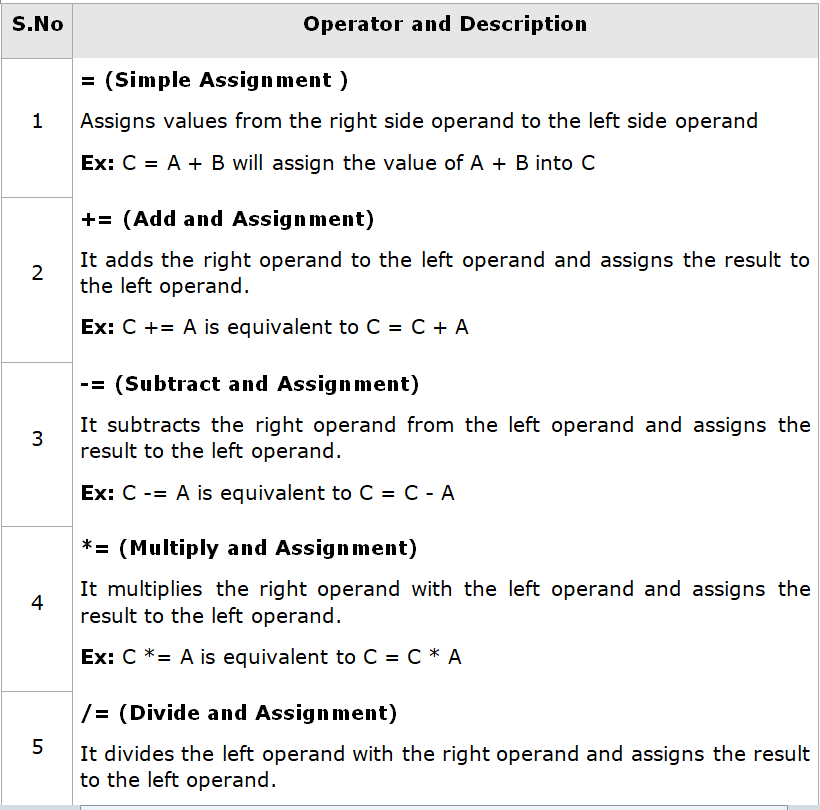
JavaScript supports the following bitwise operators: Assume variable A holds 2 and variable B holds 3, then:

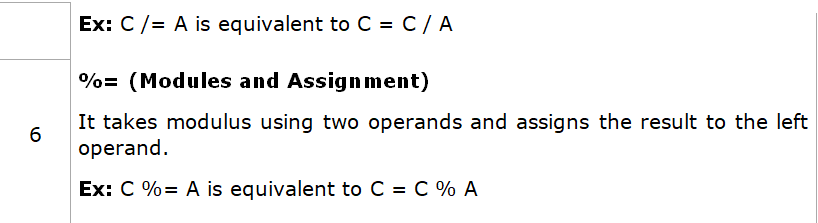
****

****

**ASSIGNMENT OPERATORS**

JavaScript supports the following assignment operators:

****

****

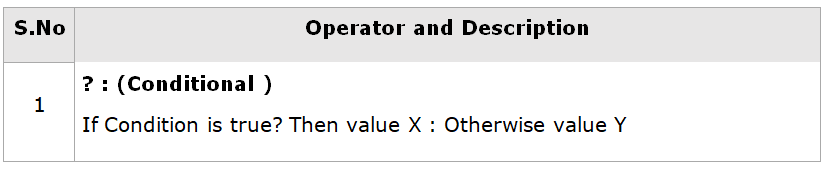
**MISCELLANEOUS OPERATORS**

We will discuss two operators here that are quite useful in JavaScript: the

##### conditional operator (? :) and the typeof operator.

### **Conditional Operator (? :)**

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

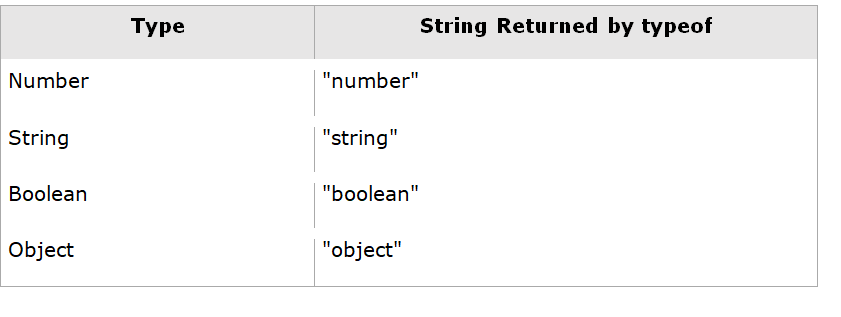


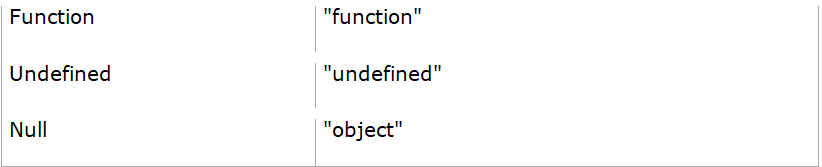
### **typeof Operator**

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.



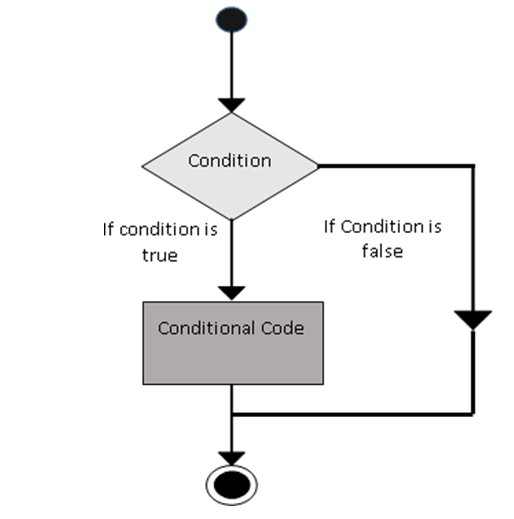


**IF-ELSE STATEMENT IN JAVASCRIPT**

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 :-



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

· if statement

· if...else statement

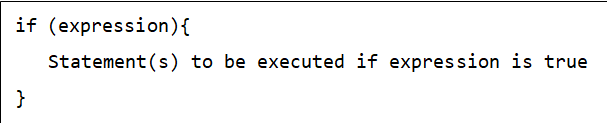
· if...else if... statement

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:



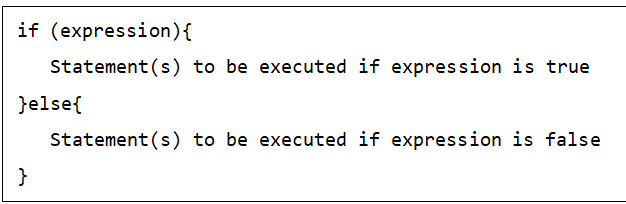
Here a JavaScript expression is evaluated. If the resulting value is true, the given statement(s) are executed. If the expression is false, then no statement would be not executed. Most of the times, you will use comparison operators while making decisions.

If-else Statement :-

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

#### Syntax

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



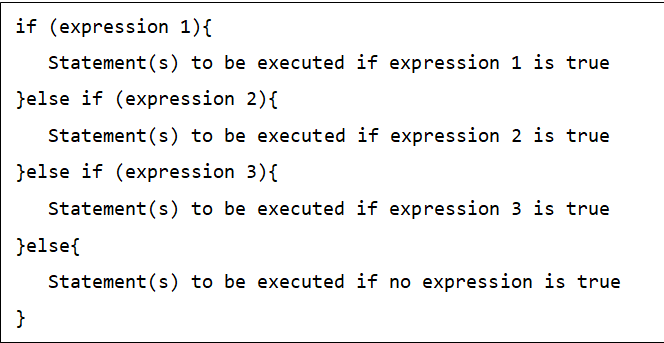
Here JavaScript expression is evaluated. If the resulting value is true, the given statement(s) in the ‘if’ block, are executed. If the expression is false, then the given statement(s) in the else block are executed.

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:



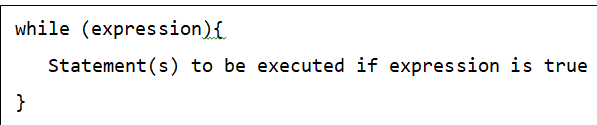
There is nothing special about this code. It is just a series of **if** statements, where each **if** is a part of the **else** clause of the previous statement. Statement(s) are executed based on the true condition, if none of the conditions is true, then the **else** block is executed.

**LOOPS IN JAVASCRIPT**

The while loop :-

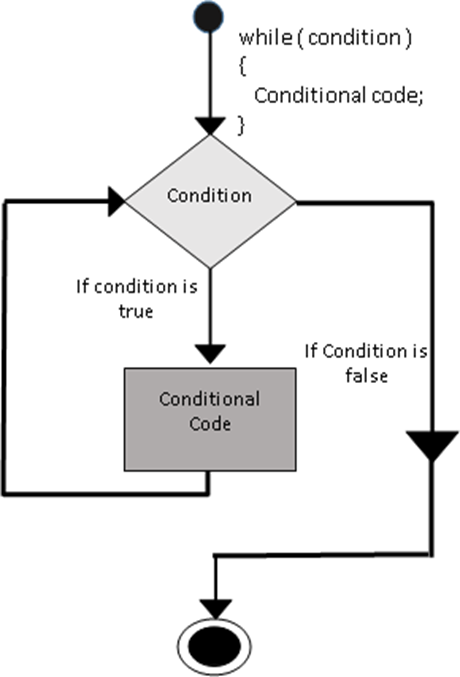
The most basic loop in JavaScript is the while loop which would be discussed in this chapter. The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates.

#### Syntax **:-**The syntax of **while loop** in JavaScript is as follows:



Flow Chart

The flow chart of while loop looks as follows:

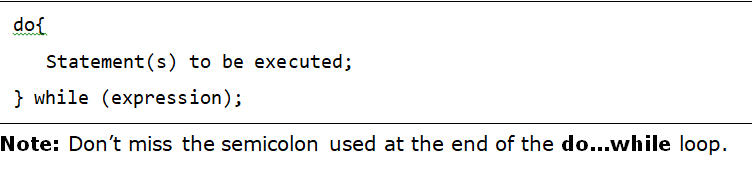


The do-while loop :-

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.

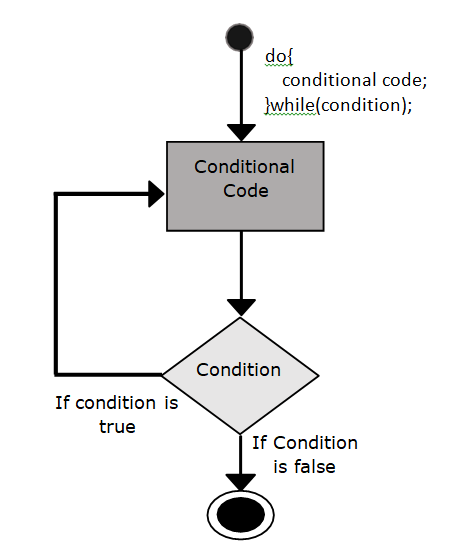
#### **Syntax**

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



#### 

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



The for loop :-

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

· The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.

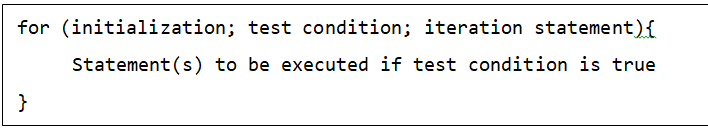
· The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.

· The iteration statement where you can increase or decrease your counter.

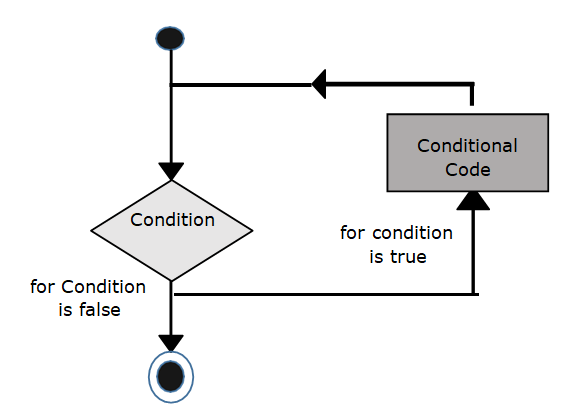
You can put all the three parts in a single line separated by semicolons.

#### **Syntax**

The syntax of **for** loop is JavaScript is as follows:

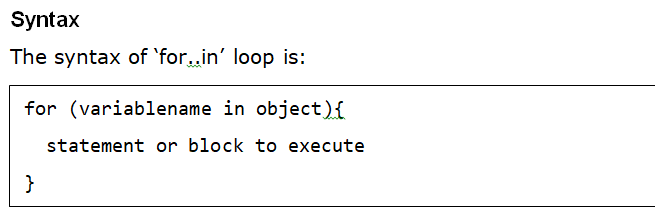


The flow chart of a **for** loop in JavaScript would be as follows:



The 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.



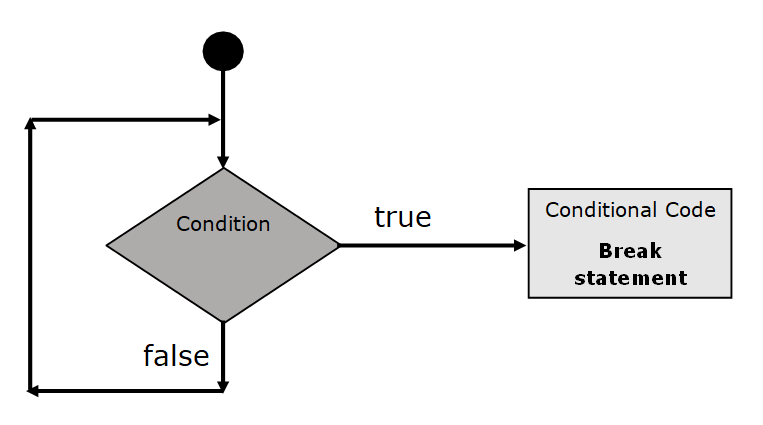
In each iteration, one property from object is assigned to variable name and this loop continues till all the properties of the object are exhausted.

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:



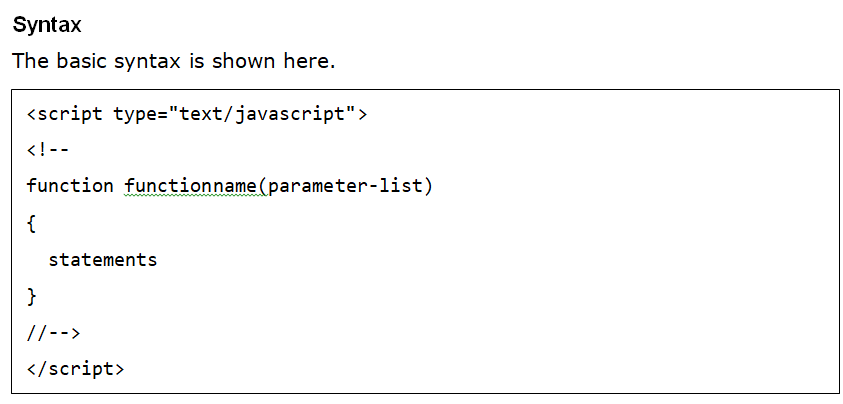
The continue statement :-

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.

**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.

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.

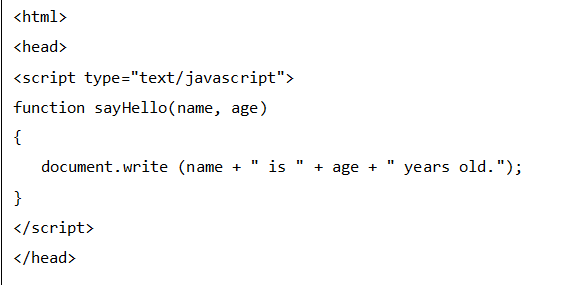


**FUNCTION PARAMETERS**

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.



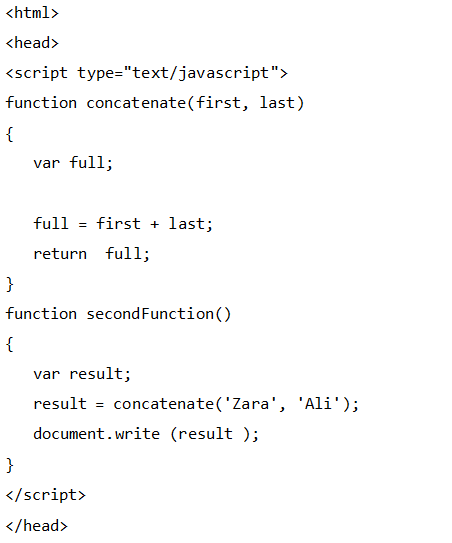
**The return Statement**

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.

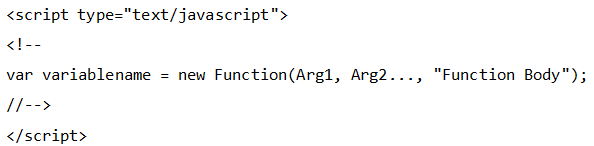
****

**FUNCTION CONSTRUCTOR**

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

#### Syntax

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



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

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

Example

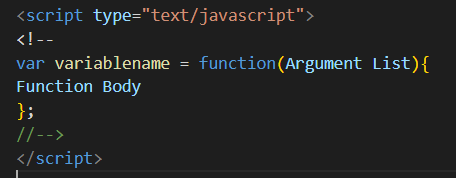


**FUNCTION LITERALS**

A function literal is an expression that defines an unnamed function.

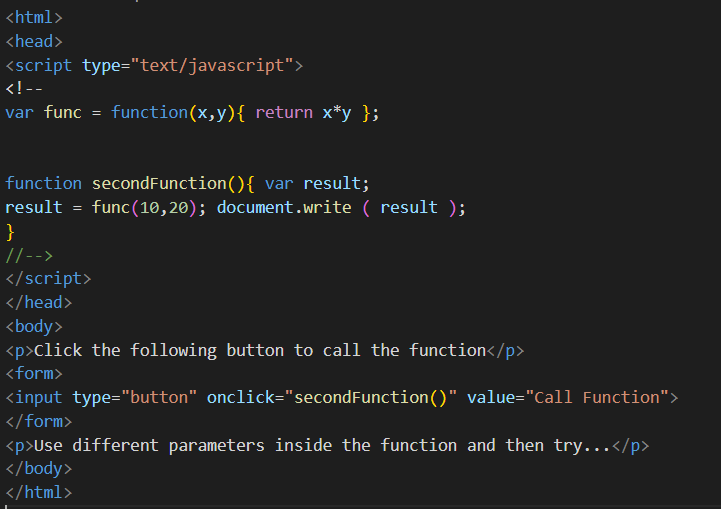
Syntax

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



#### **Example**

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



**EVENTS**

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.

Here we will see a few examples to understand the relation between Event and JavaScript.

**onclick Event Type**

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

Example

Try the following example.



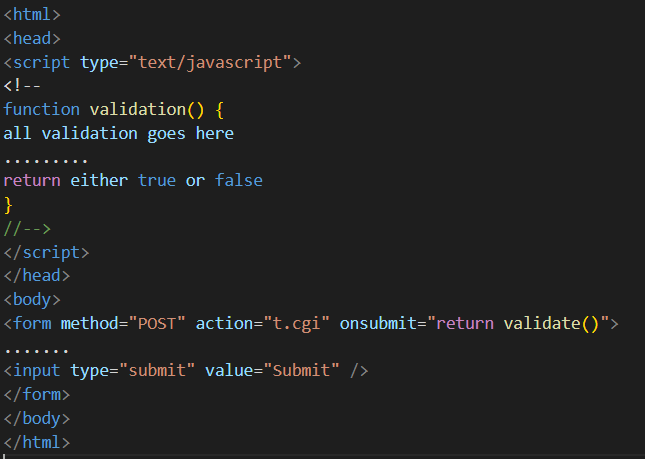
**onsubmit Event Type**

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.

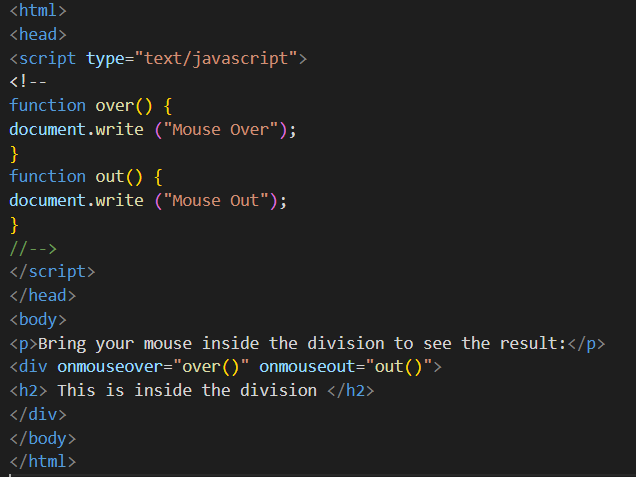
Try the following example.



**onmouseout and onmouseover**

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 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 |

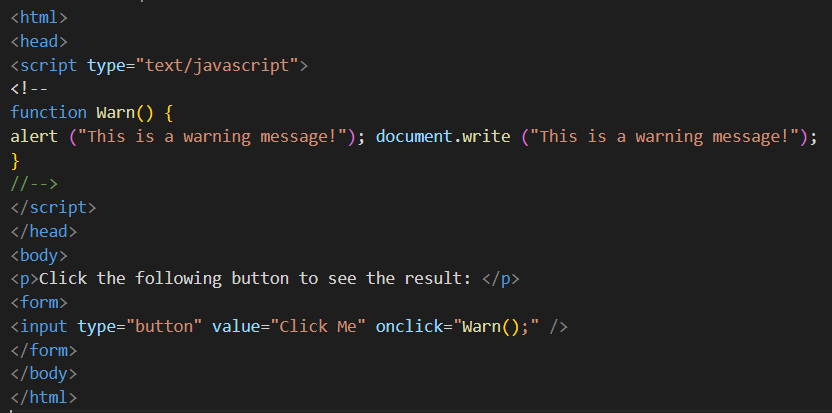
| ondragenter | **script** | Triggers when an element has been dragged to a valid drop target |
| --- | --- | --- |
| ondragleave | **script** | Triggers when an element leaves a valid drop target |
| ondragover | **script** | Triggers when an element is being dragged over a valid drop target |
| ondragstart | **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 |

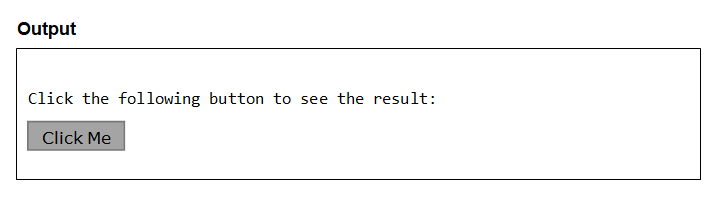
**DIALOG BOX IN JAVASCRIPT**

**ALERT DIALOG BOX**

An alert dialog box is mostly used to give a warning message to the users. For example, if one input field requires to enter some text but the user does not provide any input, then as a part of validation, you can use an alert box to give a warning message.

Nonetheless, an alert box can still be used for friendlier messages. Alert box gives only one button "OK" to select and proceed.

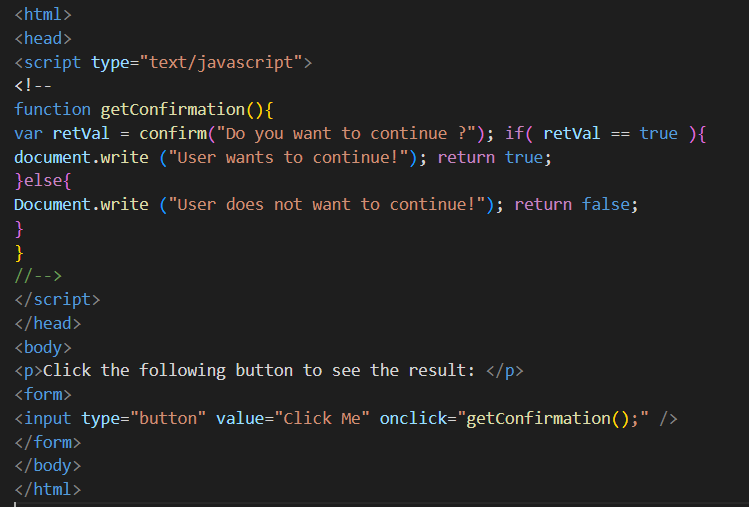


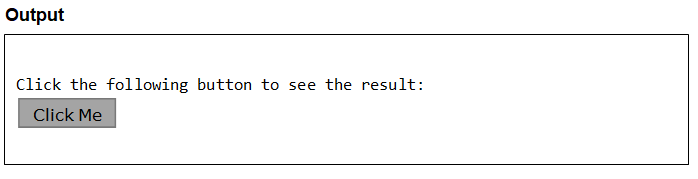
****

**Confirmation Dialog Box**

A confirmation dialog box is mostly used to take user's consent on any option. It displays a dialog box with two buttons: OK and Cancel.

If the user clicks on the OK button, the window method confirm() will return true. If the user clicks on the Cancel button, then confirm() returns false. You can use a confirmation dialog box as follows.



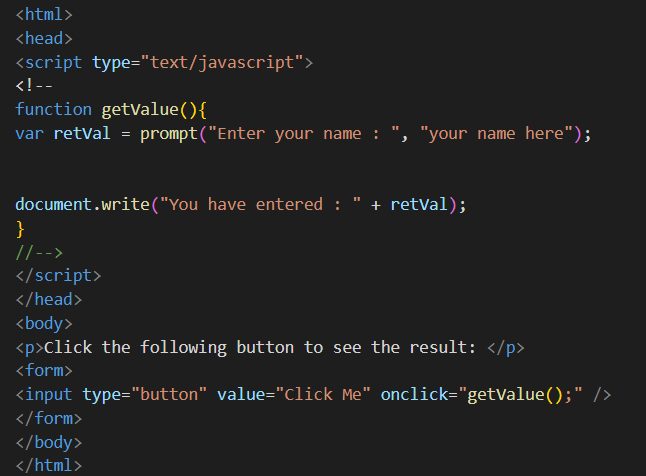
****

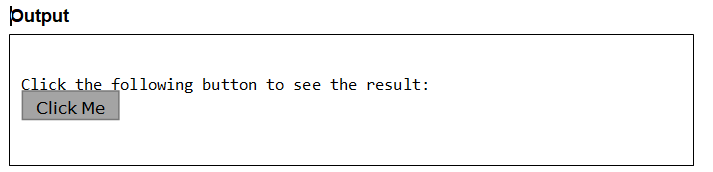
**Prompt Dialog Box**

The prompt dialog box is very useful when you want to pop-up a text box to get user input. Thus, it enables you to interact with the user. The user needs to fill in the field and then click OK.

This dialog box is displayed using a method called prompt() which takes two parameters: (i) a label which you want to display in the text box and (ii) a default string to display in the text box.

This dialog box has two buttons: OK and Cancel. If the user clicks the OK button, the window method prompt() will return the entered value from the text box. If the user clicks the Cancel button, the window method prompt() returns null.



****

**JAVASCRIPT OBJECTS**

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:



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



**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.



**USER DEFINED OBJECTS**

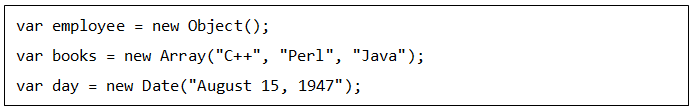
All user-defined objects and built-in objects are descendants of an object called

##### 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.



### 

### 

### 

### 

### The Object ( ) Constructor

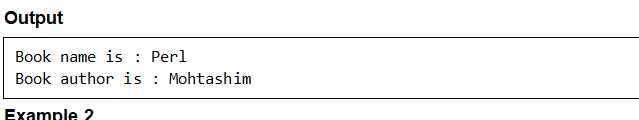
A constructor is a function that creates and initializes an object. JavaScript provides a special constructor function called **Object()** to build the object. The return value of the **Object()** constructor is assigned to a variable.

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.



****

**NUMBER METHODS**

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

**toExponential()**

This method returns a string representing the number object in exponential notation.

Syntax

Its syntax is as follows:



#### Parameter Details

fractionDigits: An integer specifying the number of digits after the decimal point. Defaults to as many digits as necessary to specify the number.

#### Return Value

A string representing a Number object in exponential notation with one digit before the decimal point, rounded to fractionDigits digits after the decimal point. If the fractionDigits argument is omitted, the number of digits after the decimal point defaults to the number of digits necessary to represent the value uniquely.

**toFixed()**

This method formats a number with a specific number of digits to the right of the decimal.

#### Syntax

Its syntax is as follows:



#### 

#### Parameter Details

digits: The number of digits to appear after the decimal point.

#### Return Value

A string representation of a numberthat does not use exponential notation and has the exact number of digits after the decimal place.

**toLocaleString()**

This method converts a number object into a human readable string representing the number using the locale of the environment.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns a human readable string representing the number using the locale of the environment.

**toPrecision()**

This method returns a string representing the number object to the specified precision.

#### Syntax

Its syntax is as follows:



#### Parameter Details

precision: An integer specifying the number of significant digits.

Return Value

Returns a string representing a Number object in fixed-point or exponential notation rounded toprecision significant digits.

**toString()**

This method returns a string representing the specified object. The toString() method parses its first argument, and attempts to return a string representation in the specified radix (base).

#### **Syntax**

Its syntax is as follows:



#### Parameter Details

radix: An integer between 2 and 36 specifying the base to use for representing numeric values.

#### Return Value

Returns a string representing the specified Number object.

**valueOf**

This method returns the primitive value of the specified number object.

#### Syntax

Its syntax is as follows:

****

#### Return Value

#### Returns the primitive value of the specified number object.

**BOOLEAN IN JAVASCRIPT**

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.



**Boolean Properties**

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

**constructor()**

Javascript boolean constructor() method returns a reference to the Boolean function that created the instance's prototype.

#### Syntax

Use the following syntax to create a Boolean constructor() method.

****

#### Return Value

Returns the function that created this object's instance.

**prototype()**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String and Date, etc.).

Note: Prototype is a global property which is available with almost all the objects.

#### Syntax

Use the following syntax to create a Boolean prototype.



**Boolean Methods**

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

**toSource()**

Javascript boolean toSource() method returns a string representing the source code of the object.

Note: This method is not compatible with all the browsers.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns a string representing the source code of the object.

**toString()**

This method returns a string of either "true" or "false" depending upon the value of the object.

#### Syntax

Its syntax is as follows:

****

#### Return Value

Returns a string representing the specified Boolean object.

**valueOf()**

Javascript boolean valueOf() method returns the primitive value of the specified boolean object.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the primitive value of the specified boolean object.

**ARRAYS IN JAVASCRIPT**

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.

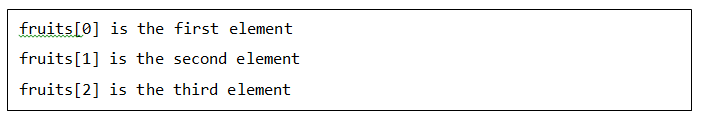
****

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:

****

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



**ARRAY PROPERTIES**

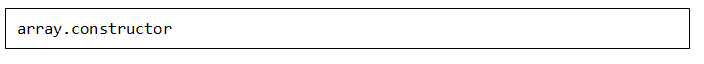
In the following sections, we will have a few examples to illustrate the usage of Array properties.

**constructor()**

Javascript array constructor property returns a reference to the array function that created the instance's prototype.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the function that created this object's instance**.**

**length()**

Javascript array length property returns an unsigned, 32-bit integer that specifies the number of elements in an array.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the length of an array.

**Prototype**

The prototype property allows you to add properties and methods to any object (Number, Boolean, String, Date, etc.).

Note: Prototype is a global property which is available with almost all the objects.

#### Syntax

Its syntax is as follows:



**ARRAY METHODS**

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

**concat()**

Javascript array concat() method returns a new array comprised of this array joined with two or more arrays.

#### Syntax

The syntax of concat() method is as follows:



#### Parameter Details

valueN : Arrays and/or values to concatenate to the resulting array.

#### Return Value

Returns the length of the array.

**every()**

Javascript array every method tests whether all the elements in an array passes the test implemented by the provided function.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· callback : Function to test for each element.

· thisObject : Object to use as this when executing callback.

#### Return Value

Returns true if every element in this array satisfies the provided testing function.

**filter()**

Javascript array filter() method creates a new array with all elements that pass the test implemented by the provided function.

#### Syntax

Its syntax is as follows:

****

#### Parameter Details

· callback : Function to test for each element of an array.

· thisObject : Object to use as this when executing callback.

#### Return Value

Returns created array.

**forEach()**

Javascript array forEach() method calls a function for each element in the array.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· callback : Function to test for each element of an array.

· thisObject : Object to use as this when executing callback.

#### Return Value

Returns the created array.

**indexOf()**

Javascript array indexOf() method returns the first index at which a given element can be found in the array, or -1 if it is not present.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· searchElement : Element to locate in the array.

· fromIndex : The index at which to begin the search. Defaults to 0, i.e. the whole array will be searched. If the index is greater than or equal to the length of the array, -1 is returned.

#### Return Value

Returns the index of the found element.

**join()**

Javascript array join() method joins all the elements of an array into a string.

#### Syntax

Its syntax is as follows:



#### Parameter Details

separator : Specifies a string to separate each element of the array. If omitted, the array elements are separated with a comma.

#### 

#### Return Value

Returns a string after joining all the array elements.

**lastIndexOf()**

Javascript array lastIndexOf() method returns the last index at which a given element can be found in the array, or -1 if it is not present. The array is searched backwards, starting at fromIndex.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· searchElement : Element to locate in the array.

· fromIndex : The index at which to start searching backwards. Defaults to the array's length, i.e., the whole array will be searched. If the index is greater than or equal to the length of the array, the whole array will be searched. If negative, it is taken as the offset from the end of the array.

#### Return Value

Returns the index of the found element from the last.

**map()**

Javascript array map() method creates a new array with the results of calling a provided function on every element in this array.

#### Syntax

Its syntax is as follows:



Parameter Details

· callback : Function that produces an element of the new Array from an element of the current one.

· thisObject : Object to use as this when executing callback.

#### Return Value

Returns the created array.

**pop()**

Javascript array pop() method removes the last element from an array and returns that element.

#### **Syntax**

Its syntax is as follows:



#### **Return Value**

Returns the removed element from the array.

**push()**

Javascript array push() method appends the given element(s) in the last of the array and returns the length of the new array.

#### Syntax

Its syntax is as follows:



#### Parameter Details

element1, ..., elementN: The elements to add to the end of the array.

#### Return Value

Returns the length of the new array.

**reduce()**

Javascript array reduce() method applies a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· callback : Function to execute on each value in the array.

· initialValue : Object to use as the first argument to the first call of the callback.

#### Return Value

Returns the reduced single value of the array.

**reduceRight()**

Javascript array reduceRight() method applies a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· callback : Function to execute on each value in the array.

· initialValue : Object to use as the first argument to the first call of the callback.

#### Return Value

Returns the reduced right single value of the array.

**reverse()**

Javascript array reverse() method reverses the element of an array. The first array element becomes the last and the last becomes the first.

#### Syntax

Its syntax is as follows:



#### Return Value

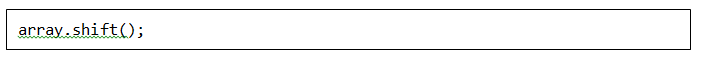
Returns the reversed single value of the array.

**shift()**

Javascript array shift() method removes the first element from an array and returns that element.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the removed single value of the array.

**slice()**

Javascript array slice() method extracts a section of an array and returns a new array.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· begin : Zero-based index at which to begin extraction. As a negative index, start indicates an offset from the end of the sequence.

· end : Zero-based index at which to end extraction.

#### Return Value

Returns the extracted array based on the passed parameters.

**some()**

Javascript array some() method tests whether some element in the array passes the test implemented by the provided function.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· callback : Function to test for each element.

· thisObject : Object to use as this when executing callback.

#### Return Value

If some element pass the test, then it returns true, otherwise false.

**sort()**

Javascript array sort() method sorts the elements of an array.

#### Syntax

Its syntax is as follows:



#### Parameter Details

compareFunction: Specifies a function that defines the sort order. If omitted, the array is sorted lexicographically.

#### Return Value

Returns a sorted array.

**splice()**

Javascript array splice() method changes the content of an array, adding new elements while removing old elements.

#### Syntax

Its syntax is as follows:



#### Parameter Details

· index: Index at which to start changing the array.

· howMany: An integer indicating the number of old array elements to remove. If howMany is 0, no elements are removed.

· element1, ..., elementN: The elements to add to the array. If you don't specify any elements, splice simply removes the elements from the array.

#### Return Value

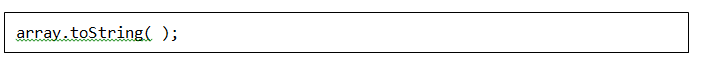
Returns the extracted array based on the passed parameters.

**toString()**

Javascript array toString() method returns a string representing the source code of the specified array and its elements.

#### Syntax

Its syntax is as follows:



Return Value

Returns a string representing the array.

**unshift()**

Javascript array unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.

#### Syntax

Its syntax is as follows:



#### Parameter Details

element1, ..., elementN : The elements to add to the front of the array.

#### Return Value

Returns the length of the new array. It returns undefined in IE browser.

**DATE IN JAVASCRIPT**

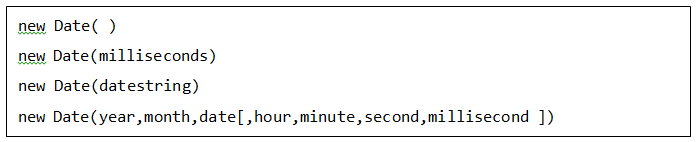
The Date object is a datatype built into the JavaScript language. Date objects are created with the new Date() as shown below.

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.



**DATE METHODS**

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

**Date()**

Javascript Date() method returns today's date and time and does not need any object to be called.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns today's date and time.

**getDate()**

Javascript date **getDate()** method returns the day of the month for the specified date according to local time. The value returned by **getDate** is an integer between 1 and 31.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns today's date and time.

**getDay()**

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

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the day of the week for the specified date according to local time.

**getFullYear()**

Javascript date getFullYear() method returns the year of the specified date according to local time. The value returned by getFullYear is an absolute number. For dates between the years 1000 and 9999, getFullYear returns a four-digit number, for example, 2008.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the year of the specified date according to local time.

**getHours()**

Javascript Date getHours() method returns the hour in the specified date according to local time. The value returned by getHours is an integer between 0 and 23.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the hour in the specified date according to local time.

**getMilliseconds()**

Javascript date getMilliseconds() method returns the milliseconds in the specified date according to local time. The value returned by getMilliseconds is a number between 0 and 999.

##### Syntax

Its syntax is as follows:



#### Return Value

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

**get Minutes()**

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

#### Syntax

Its syntax is as follows:



#### Return Value

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

**getMonth()**

Javascript date getMonth() method returns the month in the specified date according to local time. The value returned by getMonth is an integer between 0 and 11. 0 corresponds to January, 1 to February, and so on.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the Month in the specified date according to local time.

**getSeconds()**

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

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the seconds in the specified date according to local time.

**getTime()**

Javascript date getTime() method returns the numeric value corresponding to the time for the specified date according to universal time. The value returned by the getTime method is the number of milliseconds since 1 January 1970 00:00:00. You can use this method to help assign a date and time to another Date object.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the numeric value corresponding to the time for the specified date according to universal time.

**getTimezoneOffset()**

Javascript date getTimezoneOffset() method returns the time-zone offset in minutes for the current locale. The time-zone offset is the minutes in difference, the Greenwich Mean Time (GMT) is relative to your local time.

For example, if your time zone is GMT+10, -600 will be returned. Daylight savings time prevents this value from being a constant.

#### Syntax

Its syntax is as follows:



#### Return Value

Returns the time-zone offset in minutes for the current locale.

**MATH METHODS IN JAVASCRIPT**

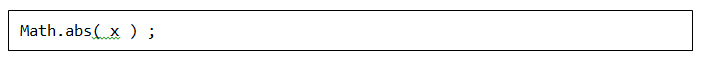
In the following sections, we will have a few examples to demonstrate the usage of the methods associated with Math.

**abs()**

This method returns the absolute value of a number.

#### Syntax

Its syntax is as follows:



#### Parameter Details

x: A number.

#### Return Value

Returns the absolute value of a number.

**acos()**

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

#### Syntax

#### Its syntax is as follows:



#### Parameter Details

x: A number.

#### Return Value

Returns the arccosine in radians of a number.

**floor()**

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

#### Syntax

Its syntax is as follows:



#### Parameter Details

x: a number.

#### Return Value

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

**random()**

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

#### Syntax

Its syntax is as follows:



#### Return Value

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

**round()**

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

#### Syntax

Its syntax is as follows:



#### Return Value

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

**sqrt()**

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

#### Syntax

Its syntax is as follows:



#### Parameter Details

x: A number.

#### Return Value

Returns the square root of a given number.

### **Document Methods in W3C DOM**

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

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

| **4**  **4** | **createElement( tagName)**  **Creates and returns a new Element node with the specified tag name.**    **Ex: document.createElement( tagName)** |
| --- | --- |
| **5**  **5** | **createTextNode( text)**  **Creates and returns a new Text node that contains the specified text.**    **Ex: document.createTextNode( text)** |
| **6**  **6** | **getElementById( id)**  **Returns the Element of this document that has the specified value for its id attribute, or null if no such Element exists in the document.**    **Ex: document.getElementById( id)** |
| **7**  **7** | **getElementsByName( name)**  **Returns an array of nodes of all elements in the document that have a specified value for their name attribute. If no such elements are found, returns a zero-length array.**    **Ex: document.getElementsByName( name)** |
| **8**    **8** | **getElementsByTagName( tagname)**  **Returns an array of all Element nodes in this document that have the specified tag name. The Element nodes appear in the returned array in the same order they appear in the document source.**    **Ex: document.getElementsByTagName( tagname)** |