

Handbook

Javascript

By: SachTech Solution Pvt. Ltd.

Introduction to Javascript

JavaScript is the Programming Language for the Web

JavaScript can update and change both HTML and CSS

JavaScript can calculate, manipulate and validate data

Type of Variables in Javascript :

JavaScript variables data type:

Number : 10 , 10.5

String : "Some Text"

Object : {obj_name:obj_value}

Array : ['value1','value2']

Function : function abc(){}

Boolean : true | false

JavaScript Arithmetic Operators:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

JavaScript Comparison Operators:

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Ternary Operator Example :

```
function getFee(isMember) {  
    return (isMember ? "$2.00" : "$10.00");  
}  
  
console.log(getFee(true));  
  
// expected output: "$2.00"  
  
console.log(getFee(false));  
  
// expected output: "$10.00"  
  
console.log(getFee(1));  
  
// expected output: "$2.00"
```

Javascript Concatenation:

+ operator: The + operator does string concatenation as soon as one of its operands is a string. Then the other operand is converted to string. Example:

```
> "Say hello " + 7 + " times fast!"
```

Output : 'Say hello 7 times fast!'

Alternatively, you can use += where

```
a += b
```

is an abbreviation for

```
a = a + b
```

JavaScript Logical Operators :

Operator	Description
&&	logical and
	logical or
!	logical not


Loops in Javascript:

JavaScript supports different kinds of loops:

for - loops through a block of code a number of times

for/in - loops through the properties of an object

while - loops through a block of code while a specified condition is true



do/while - also loops through a block of code while a specified condition is true

For Loop :

```
for (i = 0; i < 5; i++) {  
    text += "The number is " + i + "<br>";  
}
```

for/in Loop :

```
var person = {fname:"John", lname:"Doe", age:25};  
var text = "";  
var x;  
for (x in person) {  
    text += person[x];  
}
```

In the above example text variable contain all the array values as string.

While Loop :

```
Var i = 1;  
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

do/while Loop:

```
Var i = 1;  
do {
```

```
text += "The number is " + i;

i++;

}

while (i < 10);
```

JavaScript Function:

A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:

(parameter1, parameter2, ...)

The code to be executed, by the function, is placed inside curly brackets: {}

Example :

```
function name(parameter1, parameter2, parameter3) {

    code to be executed

}
```

Function Invocation :

The code inside the function will execute when "something" invokes (calls) the function:

When an event occurs (when a user clicks a button)

When it is invoked (called) from JavaScript code

Automatically (self invoked)

Function Return:



When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Functions often compute a return value. The return value is "returned" back to the "caller":

```
var x = myFunction(4, 3); // Function is called, return value will end up in x

function myFunction(a, b) {

    return a * b;        // Function returns the product of a and b

}
```

Why Functions?

You can reuse code: Define the code once, and use it many times.

You can use the same code many times with different arguments, to produce different results.

Local and Global Variable:

Local variables are declared and used inside the function only and global variables can be used anywhere in the javascript code.

```
var test = "I'm global";

function testScope() {

    var test = "I'm local";

    console.log (test);

}

testScope();        // output: I'm local

console.log(test);   // output: I'm global
```

Javascript Interval

The `setInterval()` method calls a function or evaluates an expression at specified intervals (in milliseconds).

The `setInterval()` method will continue calling the function until [clearInterval\(\)](#) is called, or the window is closed.

The ID value returned by `setInterval()` is used as the parameter for the `clearInterval()` method.

Tip: 1000 ms = 1 second.

Example :

```
var myVar = setInterval(myTimer, 1000);

function myTimer() {
    alert("hello");
}

function myStopFunction() {
    clearInterval(myVar);
}
```

Function `myTimer` will be called after every 1 second.

Function `myStopFunction` will be used to stop the interval.

What is JQuery :

Query is a lightweight, "write less, do more", JavaScript library.

The purpose of jQuery is to make it much easier to use JavaScript on your website.

jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.

jQuery Hide - Show:

```
$("#hide").click(function() {  
    $("p").hide();  
});  
  
$("#show").click(function() {  
    $("p").show();  
});
```

jQuery Fade Effect :

```
$("button").click(function() {  
    $("#div1").fadeIn();  
    $("#div2").fadeIn("slow");  
    $("#div3").fadeIn(3000);  
});  
  
$("button").click(function() {  
    $("#div1").fadeOut();  
    $("#div2").fadeOut("slow");  
    $("#div3").fadeOut(3000);
```

```
});
```

```
$("#button").click(function() {  
    $("#div1").fadeToggle();  
    $("#div2").fadeToggle("slow");  
    $("#div3").fadeToggle(3000);  
});
```

Jquery Slide:

```
$("#flip").click(function() {  
    $("#panel").slideUp();  
});  
  
$("#flip").click(function() {  
    $("#panel").slideDown();  
});  
  
$("#flip").click(function() {  
    $("#panel").slideToggle();  
});
```

Jquery CSS:

```
$("#p").css("background-color", "yellow");
```

Or if you want to use more than one style use object for the same:



```
$("p").css({background-color:'red',color:'green'});
```

Jquery Animate:

```
$("#button").click(function() {  
    $("#div").animate({left: '250px'});  
});
```

More example :

```
$("#button").click(function() {  
    $("#div").animate({  
        left: '250px',  
        opacity: '0.5',  
        height: '150px',  
        width: '150px'  
    });  
});
```

Jquery Callback:

JavaScript statements are executed line by line. However, with effects, the next line of code can be run even though the effect is not finished. This can create errors.

To prevent this, you can create a callback function.

A callback function is executed after the current effect is finished.

Typical syntax: **`$(selector).hide(speed,callback);`**

```
$("#button").click(function(){  
    $("#p").hide("slow", function(){  
        alert("The paragraph is now hidden");  
    });  
});
```

One more example:

```
$("#button").click(function(){  
    $("#p").hide(1000);  
    alert("The paragraph is now hidden");  
});
```