# **JavaScript** to program the behavior of web pages and used to make pages interactive

//Examine the document object

console.dir(document);

console.log(document.domain);

console.log(document.url);

console.log(document.doctype);

console.log(document.title);

console.log(document.all);

console.log(document.all[10]);

document.all[10].textContent = ‘Hello’;

console.log(document.forms);

console.log(document.getElementbyId(‘root’));

The <script> Tag

In HTML, JavaScript code must be inserted between <script> and </script> tags.

<!DOCTYPE html>

<html>

<body>

<h2>JavaScript in Body</h2>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = "My First JavaScript";

</script>

</body>

</html>

## JavaScript in <head> or <body>

You can place any number of scripts in an HTML document. Scripts can be placed in the <body>, or in the <head> section of an HTML page, or in both. The function is invoked (called) when a button is clicked:

<!DOCTYPE html>  
<html>

<head>  
<script>  
function myFunction() {  
    document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>  
</head>

<body>

<h1>A Web Page</h1>  
<p id="demo">A Paragraph</p>  
<button type="button" onclick="myFunction()">Try it</button>

</body>  
</html>

## JavaScript in <body>

In this example, a JavaScript function is placed in the <body> section of an HTML page. Placing scripts at the bottom of the <body> element improves the display speed, because script compilation slows down the display!!! The function is invoked (called) when a button is clicked:

<!DOCTYPE html>  
<html>  
<body>   
<h1>A Web Page</h1>  
<p id="demo">A Paragraph</p>  
<button type="button" onclick="myFunction()">Try it</button>  
<script>  
function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}  
</script>  
</body>  
</html>

## External JavaScript

Scripts can also be placed in external files: myScript.js

function myFunction() {  
   document.getElementById("demo").innerHTML = "Paragraph changed.";  
}

To use an external script, put the name of the script file in the src (source) attribute of a <script> tag:

<script src="myScript.js"></script>

**To add several script files to one page  - use several script tags:**

<script src="myScript1.js"></script>  
<script src="https://www.w3schools.com/js/myScript1.js"></script>

## JavaScript Display Possibilities

JavaScript can "display" data in different ways:

* Writing into an HTML element, using **innerHTML**.
* Writing into the HTML output using **document.write()**.The document.write() method should only be used for testing!!!
* Writing into an alert box, using **window.alert()**.
* Writing into the browser console, using **console.log()**. For debugging purposes, you can use the **console.log()** method to display data.

## JavaScript Statements

A **computer program** is a list of "instructions" to be "executed" by a computer. In a programming language, these programming instructions are called **statements**. A **JavaScript program** is a list of programming **statements**. JavaScript statements are composed of: Values, Operators, Expressions, Keywords, and Comments. The statements are executed, one by one, in the same order as they are written. JavaScript programs (and JavaScript statements) are often called JavaScript code. Semicolons separate JavaScript statements.

In JavaScript, the first character of name(identifier) must be a letter, or an underscore (\_), or a dollar sign ($). Subsequent characters may be letters, digits, underscores, or dollar signs.

**Underscore:**

first\_name, last\_name, master\_card, inter\_city.

**Upper Camel Case (Pascal Case):**

FirstName, LastName, MasterCard, InterCity.

**Lower Camel Case:**

JavaScript programmers tend to use camel case that starts with a lowercase letter: firstName, lastName, masterCard, interCity.

It is most common to use single line comments.  
Block comments are often used for formal documentation.

## JavaScript Keywords

JavaScript statements often start with a **keyword** to identify the JavaScript action to be performed.

Here is a list of some of the keywords you will learn about in this tutorial:

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| break | Terminates a switch or a loop |
| continue | Jumps out of a loop and starts at the top |
| debugger | Stops the execution of JavaScript, and calls (if available) the debugging function |
| do ... while | Executes a block of statements, and repeats the block, while a condition is true |
| for | Marks a block of statements to be executed, as long as a condition is true |
| function | Declares a function |
| if ... else | Marks a block of statements to be executed, depending on a condition |
| return | Exits a function |
| switch | Marks a block of statements to be executed, depending on different cases |
| try ... catch | Implements error handling to a block of statements |
| var | Declares a variable |

JavaScript keywords are reserved words. Reserved words cannot be used as names for variables.

## JavaScript Syntax

JavaScript syntax is the set of rules, how JavaScript programs are constructed:

var x, y;          // How to declare variables  
x = 5; y = 6;      // How to assign values  
z = x + y;         // How to compute values

## JavaScript Values

The JavaScript syntax defines two types of values: Fixed values and variable values. Fixed values are called **literals**. Variable values are called **variables**.

<p>Number can be written with or without decimals.</p>

document.getElementById("demo").innerHTML = 10.50;

<p>Strings can be written with double or single quotes.</p>

document.getElementById("demo").innerHTML = 'John Doe';

<p>In a programming language, **variables** are used to **store** data values.

JavaScript uses the **var**keyword to **declare** variables.</p>

var x; x = 5 + 6;

document.getElementById("demo").innerHTML = x;

<p>JavaScript uses arithmetic operators to compute values (just like algebra).</p> document.getElementById("demo").innerHTML = (5 + 6) \* 10;

## 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 |

## JavaScript Logical Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| && | logical and |
| || | logical or |
| ! | logical not |

## JavaScript Type Operators

|  |  |
| --- | --- |
| **Operator** | **Description** |
| typeof | Returns the type of a variable |
| instanceof | Returns true if an object is an instance of an object type |

## JavaScript Bitwise Operators

Bit operators work on 32 bits numbers.

Any numeric operand in the operation is converted into a 32 bit number. The result is converted back to a JavaScript number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Same as** | **Result** | **Decimal** |
| & | AND | 5 & 1 | 0101 & 0001 | 0001 | 1 |
| | | OR | 5 | 1 | 0101 | 0001 | 0101 | 5 |
| ~ | NOT | ~ 5 | ~0101 | 1010 | 10 |
| ^ | XOR | 5 ^ 1 | 0101 ^ 0001 | 0100 | 4 |
| << | Zero fill left shift | 5 << 1 | 0101 << 1 | 1010 | 10 |
| >> | Signed right shift | 5 >> 1 | 0101 >> 1 | 0010 | 2 |
| >>> | Zero fill right shift | 5 >>> 1 | 0101 >>> 1 | 0010 | 2 |

The examples above uses 4 bits unsigned examples. But JavaScript uses 32-bit signed numbers.  
Because of this, in JavaScript, ~ 5 will not return 10. It will return -6.  
~00000000000000000000000000000101 will return 11111111111111111111111111111010

## JavaScript Arithmetics

In arithmetic, the division of two integers produces a **quotient** and a **remainder**.  
In mathematics, the result of a **modulo operation** is the **remainder** of an arithmetic division.

## Operator Precedence (старшинство)

Multiplication (\*) and division (/) have higher **precedence** than addition (+) and subtraction (-).

var x = 100 + 50 \* 3; //250

var x = (100 + 50) \* 3; //When using parentheses, the operations inside the parentheses are computed first.

## JavaScript Operator Precedence Values

Pale red entries indicates ECMAScript 2015 (ES6) or higher.

|  |  |  |  |
| --- | --- | --- | --- |
| **Value** | **Operator** | **Description** | **Example** |
| 20 | ( ) | Expression grouping | (3 + 4) |
|  |  |  |  |
| 19 | . | Member | person.name |
| 19 | [] | Member | person["name"] |
| 19 | () | Function call | myFunction() |
| 19 | new | Create | new Date() |
|  |  |  |  |
| 17 | ++ | Postfix Increment | i++ |
| 17 | -- | Postfix Decrement | i-- |
|  |  |  |  |
| 16 | ++ | Prefix Increment | ++i |
| 16 | -- | Prefix Decrement | --i |
| 16 | ! | Logical not | !(x==y) |
| 16 | typeof | Type | typeof x |
|  |  |  |  |
| 15 | \*\* | Exponentiation (ES7) | 10 \*\* 2 |
|  |  |  |  |
| 14 | \* | Multiplication | 10 \* 5 |
| 14 | / | Division | 10 / 5 |
| 14 | % | Division Remainder | 10 % 5 |
|  |  |  |  |
| 13 | + | Addition | 10 + 5 |
| 13 | - | Subtraction | 10 - 5 |
|  |  |  |  |
| 12 | << | Shift left | x << 2 |
| 12 | >> | Shift right | x >> 2 |
| 12 | >>> | Shift right (unsigned) | x >>> 2 |
|  |  |  |  |
| 11 | < | Less than | x < y |
| 11 | <= | Less than or equal | x <= y |
| 11 | > | Greater than | x > y |
| 11 | >= | Greater than or equal | x >= y |
| 11 | in | Property in Object | "PI" in Math |
| 11 | instanceof | Instance of Object | instanceof Array |
|  |  |  |  |
| 10 | == | Equal | x == y |
| 10 | === | Strict equal | x === y |
| 10 | != | Unequal | x != y |
| 10 | !== | Strict unequal | x !== y |
|  |  |  |  |
| 9 | & | Bitwise AND | x & y |
| 8 | ^ | Bitwise XOR | x ^ y |
| 7 | | | Bitwise OR | x | y |
| 6 | && | Logical AND | x && y |
| 5 | || | Logical OR | x || y |
| 4 | ? : | Condition | ? "Yes" : "No" |
|  |  |  |  |
| 3 | += | Assignment | x += y |
| 3 | += | Assignment | x += y |
| 3 | -= | Assignment | x -= y |
| 3 | \*= | Assignment | x \*= y |
| 3 | %= | Assignment | x %= y |
| 3 | <<= | Assignment | x <<= y |
| 3 | >>= | Assignment | x >>= y |
| 3 | >>>= | Assignment | x >>>= y |
| 3 | &= | Assignment | x &= y |
| 3 | ^= | Assignment | x ^= y |
| 3 | |= | Assignment | x |= y |
|  |  |  |  |
| 2 | yield | Pause Function | yield x |
| 1 | , | Comma | 5 , 6 |

Expressions in parentheses are fully computed before the value is used in the rest of the expression.

## JavaScript Data Types

JavaScript variables can hold many **data types**: numbers, strings, objects and more:

var length = 16;                               // Number  
var lastName = "Johnson";                      // String  
var x = {firstName:"John", lastName:"Doe"};    // Object

## JavaScript Types are Dynamic

JavaScript has dynamic types. This means that the same variable can be used to hold different data types:

var x;           // Now x is undefined  
x = 5;           // Now x is a Number  
x = "John";      // Now x is a String

## JavaScript Numbers

JavaScript has only one type of numbers.Numbers can be written with, or without decimals:

var x1 = 34.00;     // Written with decimals  
var x2 = 34;        // Written without decimals

Extra large or extra small numbers can be written with scientific (exponential) notation:

var y = 123e5;      // 12300000  
var z = 123e-5;     // 0.00123

## JavaScript Arrays

JavaScript arrays are written with square brackets. Array items are separated by commas. The following code declares (creates) an array called cars, containing three items (car names):

var cars = ["Saab", "Volvo", "BMW"];

## JavaScript Objects

JavaScript objects are written with curly braces. Object properties are written as name:value pairs, separated by commas.

var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};

## Difference Between Undefined and Null

Undefined and null are equal in value but different in type:

typeof undefined           // undefined  
typeof null                // object  
  
null === undefined         // false  
null == undefined          // true

## Primitive Data

A primitive data value is a single simple data value with no additional properties and methods. The **typeof** operator can return one of these primitive types:

* string
* number
* boolean
* undefined

typeof "John"              // Returns "string"   
typeof 3.14                // Returns "number"  
typeof true                // Returns "boolean"  
typeof false               // Returns "boolean"  
typeof x                   // Returns "undefined" (if x has no value)

## Complex Data

The **typeof** operator can return one of two complex types:

* function
* object

The typeof operator returns object for both objects, arrays, and null. The typeof operator does not return object for functions.

typeof {name:'John', age:34} // Returns "object"  
typeof [1,2,3,4]            // Returns "object" (not "array", see note below)  
typeof null                  // Returns "object"  
typeof function myFunc(){}   // Returns "function"

The typeof operator returns "object" for arrays because in JavaScript arrays are objects.

## JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).

function myFunction(p1, p2) {  
    return p1 \* p2;          // The function returns the product of p1 and p2  
}

JavaScript Function Syntax

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: **{}**

function *name*(*parameter1, parameter2, parameter3*) {  
    *code to be executed*  
}

Function **parameters** are listed inside the parentheses () in the function definition. Function **arguments** are the **values** received by the function when it is invoked. Inside the function, the arguments (the parameters) behave as local variables. A Function is much the same as a Procedure or a Subroutine, in other programming languages.

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  
}

Local variables can only be accessed from within the function.

## JavaScript Objects

You have already learned that JavaScript variables are containers for data values. This code assigns a **simple value** (Fiat) to a **variable** named car:

var car = "Fiat";

Objects are variables too. But objects can contain many values. This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

var car = {type:"Fiat", model:"500", color:"white"};

The values are written as **name:value** pairs **(name and value separated by a colon).**

## Accessing Object Properties

You can access object properties in two ways:

*objectName.propertyName* or *objectName["propertyName"]*

## Object Methods

var person = {  
    firstName: "John",  
    lastName : "Doe",  
    id       : 5566,  
    fullName : function() {  
        return this.firstName + " " + this.lastName;  
    }  
};

## The ****this**** Keyword

In a function definition, **this** refers to the "owner" of the function.

In the example above, **this** is the **person object** that "owns" the **fullName** function.

In other words, **this.firstName** means the **firstName** property of **this object**.

Read more about the **this** keyword at [JS this Keyword](https://www.w3schools.com/js/js_this.asp).

## Accessing Object Methods

You access an object method with the following syntax:

*objectName.methodName()*

## Do Not Declare Strings, Numbers, and Booleans as Objects!

When a JavaScript variable is declared with the keyword "new", the variable is created as an object:

var x = new String();        // Declares x as a String object  
var y = new Number();        // Declares y as a Number object  
var z = new Boolean();       // Declares z as a Boolean object

Avoid String, Number, and Boolean objects. They complicate your code and slow down execution speed.

## JavaScript Events

Hyphens (хайсенс)- дефисы

particular task –особая задача

# **Javascript**

## Using The class Attribute in JavaScript

The class name can also be used by JavaScript to perform certain tasks for elements with the specified class name.

JavaScript can access elements with a specified class name by using the getElementsByClassName() method:

<!DOCTYPE html>

<html>

<body>

<h2>Using The class Attribute in JavaScript</h2>

<p>Click the button, to hide all elements with the class name "city", with JavaScript:</p>

<button onclick="myFunction()">Hide elements</button>

<h2 class="city">London</h2>

<p>London is the capital of England.</p>

<h2 class="city">Paris</h2>

<p>Paris is the capital of France.</p>

<h2 class="city">Tokyo</h2>

<p>Tokyo is the capital of Japan.</p>

<script>

function myFunction() {

var x = document.getElementsByClassName("city");

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

x[i].style.display = "none";

}

}

</script>

</body>

</html>

## Using The id Attribute in JavaScript

JavaScript can access an element with a specified id by using the getElementById() method:

<!DOCTYPE html>

<html>

<body>

<h2>Using The id Attribute in JavaScript</h2>

<p>JavaScript can access an element with a specified id by using the getElementById() method:</p>

<h1 id="myHeader">Hello World!</h1>

<button onclick="displayResult()">Change text</button>

<script>

function displayResult() {

document.getElementById("myHeader").innerHTML = "Have a nice day!";

}

</script>

</body>

</html>

<!DOCTYPE html>

<html>

<body>

<h1>My First JavaScript</h1>

<button type="button"

onclick="document.getElementById('demo').innerHTML = Date()">

Click me to display Date and Time.</button>

<p id="demo"></p>

</body>

</html>

## The HTML <script> Tag

The <script> tag is used to define a client-side script (JavaScript).

The <script> element either contains scripting statements, or it points to an external script file through the srcattribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

To select an HTML element, JavaScript very often uses the document.getElementById() method.

This JavaScript example writes "Hello JavaScript!" into an HTML element with id="demo":

<!DOCTYPE html>

<html>

<body>

<h2>Use JavaScript to Change Text</h2>

<p>This example writes "Hello JavaScript!" into an HTML element with id="demo":</p>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>

</body>

</html>

## A Taste of JavaScript

Here are some examples of what JavaScript can do:

### **JavaScript can change HTML content**

<!DOCTYPE html>

<html>

<body>

<h1>My First JavaScript</h1>

<p>JavaScript can change the content of an HTML element:</p>

<button type="button" onclick="myFunction()">Click Me!</button>

<p id="demo">This is a demonstration.</p>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Hello JavaScript!";

}

</script>

</body>

</html>

### **JavaScript can change HTML styles**

<!DOCTYPE html>

<html>

<body>

<h1>My First JavaScript</h1>

<p id="demo">JavaScript can change the style of an HTML element.</p>

<script>

function myFunction() {

document.getElementById("demo").style.fontSize = "25px";

document.getElementById("demo").style.color = "red";

document.getElementById("demo").style.backgroundColor = "yellow";

}

</script>

<button type="button" onclick="myFunction()">Click Me!</button>

</body>

</html>

### **JavaScript can change HTML attributes**

<!DOCTYPE html>

<html>

<body>

<script>

function light(sw) {

var pic;

if (sw == 0) {

pic = "pic\_bulboff.gif"

} else {

pic = "pic\_bulbon.gif"

}

document.getElementById('myImage').src = pic;

}

</script>

<img id="myImage" src="pic\_bulboff.gif" width="100" height="180">

<p>

<button type="button" onclick="light(1)">Light On</button>

<button type="button" onclick="light(0)">Light Off</button>

</p>

</body>

</html>

## The HTML <noscript> Tag

The <noscript> tag is used to provide an alternate content for users that have disabled scripts in their browser or have a browser that doesn't support client-side scripts:

<!DOCTYPE html>

<html>

<body>

<p id="demo"></p>

<script>

document.getElementById("demo").innerHTML = "Hello JavaScript!";

</script>

<noscript>Sorry, your browser does not support JavaScript!</noscript>

<p>A browser without support for JavaScript will show the text written inside the noscript element.</p>

</body>

</html>

## The HTML <script> Element

The <script> element is used to define client-side JavaScripts.

This JavaScript writes "Hello JavaScript!" into an HTML element with id="demo":

<!DOCTYPE html>

<html>

<head>

<title>Page Title</title>

<script>

function myFunction() {

document.getElementById("demo").innerHTML = "Hello JavaScript!";

}

</script>

</head>

<body>

<h1>My Web Page</h1>

<p id="demo">A Paragraph</p>

<button type="button" onclick="myFunction()">Try it</button>

</body>

</html>