## **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  var x = y + z;  <script>  var x, y;  x = 5 + 6;  y = x \* 10;  document.getElementById("demo").innerHTML = y;  </script> |

## **JavaScript Comparison Operators**

|  |  |
| --- | --- |
| **Operator** | **Description** |
| == | equal to |
| **===** | **equal value and equal type** |
| != | not equal |
| **!==** | **not equal value or not equal type** |
| ? | 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 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() |
| 16 | ++ | Prefix Increment | ++i |
| 16 | -- | Prefix Decrement | --i |
| 16 | ! | Logical not | !(x==y) |
| 16 | typeof | Type | typeof x |
| 14 | % | Division Remainder | 10 % 5 |
| 12 | << | Shift left | x << 2 |
| 12 | >> | Shift right | x >> 2 |
| 12 | >>> | Shift right (unsigned) | x >>> 2 |
| 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 |
| 2 | yield | Pause Function | yield x |
| 1 | , | Comma | 5 , 6 |

## JavaScript Assignment Operators

Assignment operators assign values to JavaScript variables.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example** | **Same As** |
| <<= | x <<= y | x = x << y |
| >>= | x >>= y | x = x >> y |
| >>>= | x >>>= y | x = x >>> y |
| &= | x &= y | x = x & y |
| ^= | x ^= y | x = x ^ y |
| |= | x |= y | x = x | y |
| \*\*= | x \*\*= y | x = x \*\* y |

The \*\*= operator is an experimental part of the ECMAScript 2016 proposal (ES7). It is not stable across browsers. Do not use it.

# JavaScript Data Types

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

var x = 16 + 4 + "Volvo"; //20Volvo

var x = "Volvo" + 16 + 4; // Volvo164

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

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

Strings are written with quotes. You can use single or double quotes:

var carName1 = "Volvo XC60";   // Using double quotes  
var carName2 = 'Volvo XC60';   // Using single quotes

<script>

var answer1 = "It's alright";

var answer2 = "He is called 'Johnny'";

var answer3 = 'He is called "Johnny"';

document.getElementById("demo").innerHTML = answer1 + "<br>" + answer2 + "<br>" + answer3;

</script>

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 Booleans

Booleans can only have two values: true or false.

var x = 5;  
var y = 5;  
var z = 6;  
(x == y)       // Returns true  
(x == z)       // Returns false

<script>

var x = 5;

var y = 5;

var z = 6;

document.getElementById("demo").innerHTML =

(x == y) + "<br>" + (x == z);

</script>

## JavaScript Arrays

<script>

var cars = ["Saab", "Volvo", "BMW"];

document.getElementById("demo").innerHTML = cars[0];

</script>

## JavaScript Objects

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

document.getElementById("demo").innerHTML = person.firstName + " is " + person.age + " years old.";

## The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

typeof ""             // Returns "string"  
typeof "John"         // Returns "string"  
typeof "John Doe"     // Returns "string"

<script>

document.getElementById("demo").innerHTML = typeof "" + "<br>" + typeof "John" + "<br>" + typeof "John Doe";

</script>

typeof 0              // Returns "number"  
typeof 314            // Returns "number"  
typeof 3.14           // Returns "number"  
typeof (3)            // Returns "number"  
typeof (3 + 4)        // Returns "number"

<script>

document.getElementById("demo").innerHTML =

typeof 0 + "<br>" + typeof 314 + "<br>" + typeof 3.14 + "<br>" + typeof (3) + "<br>" + typeof (3 + 4);

</script>

## Undefined

<script>

var car;    // Value is undefined, type is undefined

document.getElementById("demo").innerHTML = car + "<br>" + typeof car;

</script>

Any variable can be emptied, by setting the value to undefined. The type will also be undefined.

car = undefined;    // Value is undefined, type is undefined

<script>

var car = "Volvo";

car = undefined;

document.getElementById("demo").innerHTML = car + "<br>" + typeof car;

</script>

## Empty Values

var car = "";    // The value is "", the typeof is "string"

<script>

var car = "";

document.getElementById("demo").innerHTML = "The value is: " + car + "<br>" + "The type is: " + typeof car;

</script>

## Null

In JavaScript null is "nothing". It is supposed to be something that doesn't exist. Unfortunately, in JavaScript, the data type of null is an object. You can consider it a bug in JavaScript that typeof null is an object. It should be null.

You can empty an object by setting it to null:

<script>

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

person = null;

document.getElementById("demo").innerHTML = typeof person; // object

</script>

You can also empty an object by setting it to undefined:

<script>

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

person = undefined;

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

</script>

## 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

<script>

document.getElementById("demo").innerHTML = typeof undefined + "<br>" + typeof null + "<br><br>" + (null === undefined) + "<br>" + (null == undefined);

</script>

## Primitive Data

A primitive data value is a single simple data value with no additional properties and methods.

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)

<script>

document.getElementById("demo").innerHTML =

typeof "john" + "<br>" +

typeof 3.14 + "<br>" +

typeof true + "<br>" +

typeof false + "<br>" +

typeof x;

</script>

## **Complex Data**

The typeof operator can return one of two complex types: function or 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"

<script>

document.getElementById("demo").innerHTML =

typeof {name:'john', age:34} + "<br>" +

typeof [1,2,3,4] + "<br>" +

typeof null + "<br>" +

typeof function myFunc(){};

</script>

# JavaScript Functions

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

**Example:**

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

<script>

function myFunction(p1, p2) {

return p1 \* p2;

}

document.getElementById("demo").innerHTML = myFunction(4, 3);

</script>

## Function Return

<script>

var x = myFunction(4, 3);

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

**function myFunction(a, b) {**

**return a \* b;**

**}**

</script>

Convert Fahrenheit to Celsius:

<script>

function toCelsius(f) {

return (5/9) \* (f-32);

}

document.getElementById("demo").innerHTML = toCelsius(77);

</script>

Accessing a function without () will return the function definition instead of the function result:

function toCelsius(fahrenheit) {  
  return (5/9) \* (fahrenheit-32);  
}  
document.getElementById("demo").innerHTML = toCelsius; // function toCelsius(f) { return (5/9) \* (f-32); }

## Functions Used as Variable Values

var x = toCelsius(77);  
var text = "The temperature is " + x + " Celsius";

var text = "The temperature is " + toCelsius(77) + " Celsius"; //same as above

## Local Variables

Variables declared within a JavaScript function, become **LOCAL** to the function. Local variables can only be accessed from within the function.

<script>

myFunction();

function myFunction() {

var carName = "Volvo";

document.getElementById("demo1").innerHTML = typeof carName + " " + carName; // string Volvo

}

document.getElementById("demo2").innerHTML = typeof carName; // undefined

</script>

# JavaScript Objects

object: car

properties: car.name

method: car.start() car.drive() car.stop()

https://www.w3schools.com/js/js\_objects.asp