JavaScript Operators Reference

‹ Previous Next ›

JavaScript Operators

Operators are used to assign values, compare values, perform arithmetic operations, and more.

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators
- Conditional Operators
- Type Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values.

Given that **y = 5**, the table below explains the arithmetic operators:

Oper	Name	Example	Results	Try it
+	Addition	x = y + 2	y=5, x=7	Try it »
-	Subtraction	x=y-2	y=5, x=3	Try it »
*	Multiplication	x=y*2	y=5, x=10	Try it »
**	Exponentiation ES2016	x=y**2	y=5, x=25	Try it »
/	Division	x = y / 2	y=5, x=2.5	Try it »

%	Remainder	x = y % 2	y=5, x=1	Try it »
++	Pre increment	x = ++y	y=6, x=6	Try it »
++	Post increment	x = y++	y=6, x=5	Try it »
--	Pre decrement	x = --y	y=4, x=4	Try it »
--	Post decrement	x = y--	y=4, x=5	Try it »

For a tutorial about arithmetic operators, read our JavaScript Arithmetic Tutorial.

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables.

Given that **x = 10** and **y = 5**, the table below explains the assignment operators:

Oper	Example	Same As	Result	Try it
=	x = y	x = y	x = 5	Try it »
+=	x += y	x = x + y	x = 15	Try it »
-=	x -= y	x = x - y	x = 5	Try it »
*=	x *= y	x = x * y	x = 50	Try it »
/=	x /= y	x = x / y	x = 2	Try it »
%=	x %= y	x = x % y	x = 0	Try it »
:	x: 45	size.x = 45	x = 45	Try it »

For a tutorial about assignment operators, read our JavaScript Assignment Tutorial.

JavaScript String Operators

The + operator, and the += operator can also be used to concatenate (add) strings.

Given that **t1 = "Good "**, **t2 = "Morning"**, **and t3 = ""**, the table below explains the operators:

Oper	Example	t1	t2	t3	Try it
+	t3 = t1 + t2	"Good "	"Morning"	"Good Morning"	Try it »
+=	t1 += t2	"Good Morning"	"Morning"		Try it »

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that **x = 5**, the table below explains the comparison operators:

Oper	Name	Comparing	Returns	Try it
==	equal to	x == 8	false	Try it »

==	equal to	x == 5	true	Try it »
===	equal value and type	x === "5"	false	Try it »
===	equal value and type	x === 5	true	Try it »
!=	not equal	x != 8	true	Try it »
!==	not equal value or type	x !== "5"	true	Try it »
!==	not equal value or type	x !== 5	false	Try it »
>	greater than	x > 8	false	Try it »
<	less than	x < 8	true	Try it »
>=	greater or equal to	x >= 8	false	Try it »
<=	less or equal to	x <= 8	*true*	Try it »

For a tutorial about comparison operators, read our JavaScript Comparisons Tutorial.

Conditional (Ternary) Operator

The conditional operator assigns a value to a variable based on a condition.

Syntax	**Example**	**Try it**
(condition) ? x : y	(z < 18) ? x : y	Try it »

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that **x = 6 and y = 3**, the table below explains the logical

operators:

Oper	Name	Example	Try it
&&	AND	(x < 10 && y > 1) is true	Try it »
||	OR	(x === 5 || y === 5) is false	Try it »
!	NOT	!(x === y) is true	Try it »

The Nullish Coalescing Operator (??)

The `??` operator returns the first argument if it is not **nullish** (`null` or `undefined`).

Otherwise it returns the second argument.

Example

```
let name = null;
let text = "missing";
let result = name ?? text;
```

Try it Yourself »

The nullish operator is supported in all browsers since March 2020:

Chrome 80	Edge 80	Firefox 72	Safari 13.1	Opera 67
Feb 2020	Feb 2020	Jan 2020	Mar 2020	Mar 2020

The Optional Chaining Operator (?.)

The `?.` operator returns `undefined` if an object is `undefined` or

`null` (instead of throwing an error).

Example

```
const car = {type:"Fiat", model:"500", color:"white"};
document.getElementById("demo").innerHTML = car?.name;
```

Try it Yourself »

The optional chaining operator is supported in all browsers since March 2020:

Chrome 80	Edge 80	Firefox 72	Safari 13.1	Opera 67
Feb 2020	Feb 2020	Jan 2020	Mar 2020	Mar 2020

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.

Oper	Name	Example	Same as	Result	Decimal	Try it
&	AND	x = 5 & 1	0101 & 0001	0001	1	Try it »
|	OR	x = 5 | 1	0101 | 0001	0101	5	Try it »
~	NOT	x = ~ 5	~0101	1010	10	Try it »
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4	Try it »

<<	Left shift	x = 5 << 1	0101 << 1	1010	10	Try it »
>>	Right shift	x = 5 >> 1	0101 >> 1	0010	2	Try it »
>>>	Unsigned right	x = 5 >>> 1	0101 >>> 1	0010	2	Try it »

Note

The table above uses 4 bits unsigned number. Since JavaScript uses 32-bit signed numbers, ~ 5 will not return 10. It will return -6.

~00000000000000000000000000000101 (~5)
will return
11111111111111111111111111111010 (-6)

The typeof Operator

The **typeof** operator returns the type of a variable, object, function or expression:

Example

```
typeof "John"    typeof 3.14
```

Try it Yourself »

Please observe:

- The data type of NaN is number
- The data type of an array is object
- The data type of a date is object
- The data type of null is object

- The data type of an undefined variable is undefined

Example

```
typeof "John"
typeof 3.14
typeof NaN
typeof false
typeof [1, 2, 3, 4]
typeof {name:'John', age:34}
typeof new Date()
typeof function () {}
typeof myCar
typeof null
```

Try it Yourself »

Note

You cannot use **typeof** to define if a JavaScript object is an array or a date.

Both array and date return object as type.

The delete Operator

The **delete** operator deletes a property from an object:

Example

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

```
delete person.age;
```

Try it Yourself »

The delete operator deletes both the value of the property and the property itself.

After deletion, the property cannot be used before it is added back again.

The delete operator is designed to be used on object properties. It has no effect on variables or functions.

Note

The delete operator should not be used on the properties of any predefined JavaScript objects (Array, Boolean, Date, Function, Math, Number, RegExp, and String).

This can crash your application.

The Spread (...) Operator

The ... operator expands an iterable into more elements:

Example

```
const q1 = ["Jan", "Feb", "Mar"];
const q2 = ["Apr", "May", "Jun"];
const q3 = ["Jul", "Aug", "Sep"];
const q4 = ["Oct", "Nov", "May"];

const year = [...q1, ...q2, ...q3, ...q4];
```

Try it Yourself »

The ... operator can be used to expand an iterable into more arguments for function calls:

Example

```
const numbers = [23,55,21,87,56];
let maxValue = Math.max(...numbers);
```

Try it Yourself »

The in Operator

The **in** operator returns true if a property is in an object, otherwise false:

Object Example

```
const person = {firstName:"John", lastName:"Doe", age:50};
("firstName" in person);
("age" in person);
```

Try it Yourself »

Note

You cannot use **in** to check for array content like ("Volvo" in cars).

Array properties can only be index (0,1,2,3...) and length.

See the examples below.

Examples

```
const cars = ["Saab", "Volvo", "BMW"];
("Saab" in cars);
```

Try it Yourself »

```
const cars = ["Saab", "Volvo", "BMW"];
(0 in cars);
(1 in cars);
(4 in cars);
("length" in cars);
```

Try it Yourself »

Predefined Objects

```
("PI" in Math);
("NaN" in Number);
("length" in String);
```

Try it Yourself »

The instanceof Operator

The **instanceof** operator returns true if an object is an instance of a specified object:

Example

```
const cars = ["Saab", "Volvo", "BMW"];

(cars instanceof Array)    (cars instanceof Object)   (cars instanceof
String)   (cars instanceof Number)
```

Try it Yourself »

The void Operator

The **void** operator evaluates an expression and returns **undefined**. This operator is often used to obtain the undefined primitive value, using "void(0)" (useful when evaluating an expression without using the return value).

Example

```
<a href="javascript:void(0);">
  Useless link
</a>

<a href="javascript:void(document.body.style.backgroundColor='red');">
  Click me to change the background color of body to red
</a>
```

Try it Yourself »

See Also:

JavaScript Operator Precedence

‹ Previous Next ›