JavaScript Arithmetic Operators

**Summary**: in this tutorial, you will learn how to use JavaScript arithmetic operators to perform arithmetic calculations.

Introduction to the JavaScript Arithmetic Operators

JavaScript supports the following standard arithmetic operators:

| **Operator** | **Sign** |
| --- | --- |
| Addition | + |
| Subtraction | - |
| Multiplication | \* |
| Division | / |

An arithmetic operator accepts numerical values as operands and returns a single numerical value. The numerical values can be literals or variables.

Addition operator (+)

The addition operator returns the sum of two values. For example, the following uses the addition operator to calculate the sum of two numbers:

let sum = 10 + 20;

console.log(sum); *// 30*

Code language: JavaScript (javascript)

Also, you can use the addition operator with two [variables](https://www.javascripttutorial.net/javascript-variables/). For example:

let netPrice = 9.99,

shippingFee = 1.99;

let grossPrice = netPrice + shippingFee;

console.log(grossPrice);

Code language: JavaScript (javascript)

Output:

11.98

Code language: CSS (css)

If either value is a [string](https://www.javascripttutorial.net/string/), the addition operator uses the following rules:

* If both values are strings, it concatenates the second string to the first one.
* If one value is a string, it implicitly converts the numeric value into a string and concatenates two strings.

For example, the following uses the addition operator to add concatenate two strings:

let x = '10',

y = '20';

let result = x + y;

console.log(result);

Code language: JavaScript (javascript)

Output:

1020

The following example shows how to use the addition operator to calculate the sum of a number and a string:

let result = 10 + '20';

console.log(result);

Code language: JavaScript (javascript)

Output:

1020

In this example, JavaScript converts the number 10 into a string '10' and concatenates the second string '20' to it.

The following table shows the result when using the addition operator with special numbers:

| **First Value** | **Second Value** | **Result** | **Explanation** |
| --- | --- | --- | --- |
| NaN |  | NaN | If either value is NaN, the result is NaN |
| Infinity | Infinity | Infinity | Infinity + Infinity = NaN |
| -Infinity | -Infinity | -Infinity | -Infinity + ( -Infinity) = – Infinity |
| Infinity | -Infinity | NaN | Infinity + -Infinity = NaN |
| +0 | +0 | +0 | +0 + (+0) = +0 |
| -0 | +0 | +0 | -0 + (+0) = +0 |
| -0 | -0 | -0 | -0 + (-0) = -0 |

Subtraction operator (-)

The subtraction operator (-) subtracts one number from another. For example:

let result = 30 - 10;

console.log(result); *// 20*

Code language: JavaScript (javascript)

If a value is a string, a boolean, null, or undefined, the JavaScript engine will:

* First, convert the value to a number using the Number() function.
* Second, perform the subtraction.

The following table shows how to use the subtraction operator with special values:

| **First Value** | **Second Value** | **Result** | **Explanation** |
| --- | --- | --- | --- |
| NaN |  | NaN | If either value is NaN, the result is NaN |
| Infinity | Infinity | NaN | Infinity – Infinity = Infinity |
| -Infinity | -Infinity | -Infinity | -Infinity – ( -Infinity) = -Infinity |
| Infinity | -Infinity | Infinity | Infinity |
| +0 | +0 | +0 | +0 – (+0) = +0 |
| +0 | -0 | -0 | +0 – (-0) = -0 |
| -0 | -0 | +0 | -0 – (-0) = +0 |

Multiplication operator (\*)

JavaScript uses the asterisk (\*) to represent the multiplication operator. The multiplication operator multiplies two numbers and returns a single value. For example:

let result = 2 \* 3;

console.log(result);

Code language: JavaScript (javascript)

Output:

6

If either value is not a number, the JavaScript engine implicitly converts it into a number using the Number() function and performs the multiplication. For example:

let result = '5' \* 2;

console.log(result);

Code language: JavaScript (javascript)

Output:

10

The following table shows how the multiply operator behaves with special values:

| **First Value** | **Second Value** | **Result** | **Explanation** |
| --- | --- | --- | --- |
| NaN |  | NaN | If either value is NaN, the result is NaN |
| Infinity | 0 | NaN | Infinity \* 0 = NaN |
| Infinity | Positive number | Infinity | -Infinity \* 100 = -Infinity |
| Infinity | Negative number | -Infinity | Infinity \* (-100) = Infinity |
| Infinity | Infinity | Infinity | Infinity \* Infinity = Infinity |

Divide operator (/)

Javascript uses the slash (/) character to represent the divide operator. The divide operator divides the first value by the second one. For example:

let result = 20 / 10;

console.log(result); *// 2*

Code language: JavaScript (javascript)

If either value is not a number, the JavaScript engine converts it into a number for division. For example:

let result = '20' / 2;

console.log(result); *// 10;*

Code language: JavaScript (javascript)

The following table shows the divide operators’ behavior when applying to special values:

| **First Value** | **Second Value** | **Result** | **Explanation** |
| --- | --- | --- | --- |
| NaN |  | NaN | If either value is NaN, the result is NaN |
| A number | 0 | Infinity | 1/0 = Infinity |
| Infinity | Infinity | NaN | Infinity / Infinity = NaN |
| 0 | 0 | NaN | 0/0 = NaN |
| Infinity | A positive number | Infinity | Infinity / 2 = Infinity |
| Infinity | A negative number | -Infinity | Infinity / -2 = -Infinity |

Using JavaScript arithmetic operators with objects

If a value is an [object](https://www.javascripttutorial.net/javascript-objects/), the JavaScript engine will call the valueOf() method of the object to get the value for calculation. For example:

let energy = {

valueOf() {

return 100;

},

};

let currentEnergy = energy - 10;

console.log(currentEnergy);

currentEnergy = energy + 100;

console.log(currentEnergy);

currentEnergy = energy / 2;

console.log(currentEnergy);

currentEnergy = energy \* 1.5;

console.log(currentEnergy);

Code language: JavaScript (javascript)

Output:

90

200

50

150

If the object doesn’t have the valueOf() method but has the toString() method, the JavaScript engine will call the toString() method to get the value for calculation. For example:

let energy = {

toString() {

return 50;

},

};

let currentEnergy = energy - 10;

console.log(currentEnergy);

currentEnergy = energy + 100;

console.log(currentEnergy);

currentEnergy = energy / 2;

console.log(currentEnergy);

currentEnergy = energy \* 1.5;

console.log(currentEnergy);

Code language: JavaScript (javascript)

Output:

40

150

25

75

Summary

* Use the JavaScript arithmetic operators including addition (+), subtraction (-), multiply (\*) and divide (/) to perform arithmetic operations.

# JavaScript Remainder Operator

**Summary**: in this tutorial, you’ll learn about the JavaScript remainder operator (%) to get the remainder of a number divided by another number.

## Introduction to the JavaScript remainder operator

JavaScript uses the % to represent the remainder operator. The remainder operator returns the remainder left over when one value is divided by another value.

Here’s the syntax of the remainder operator:

dividend % divisor

The following shows the equation for the remainder:

dividend = divisor \* quotient + remainder

where |remainder| < |divisor|

In this equation, the dividend, divisor, quotient, and remainder are all integers. The sign of the remainder is the same as the sign of the dividend.

The sign of the remainder is the same as the sign of the dividend.

## JavaScript remainder operator examples

Let’s take some examples of using the JavaScript remainder operator.

### **1) Using the remainder operator with positive dividend example**

The following example shows how to use the remainder operator with a positive dividend:

let remainder = 5 % -2;

console.log(remainder); *// 1*

remainder = 5 % 2;

console.log(remainder); *// 1*

Code language: JavaScript (javascript)

### **2) Using the remainder operator with negative dividend example**

The following example uses the remainder operator with a negative dividend:

let remainder = -5 % 3;

console.log(remainder); *// -2*

remainder = -5 % -3;

console.log(remainder); *// -2*

Code language: JavaScript (javascript)

### **3) Using the remainder operator special values**

If a dividend is an Infinity and a divisor is a finite number, the remainder is NaN. For example:

let remainder = Infinity % 2;

console.log(remainder); *// NaN*

Code language: JavaScript (javascript)

If a dividend is a finite number and a divisor is zero, the remainder is NaN:

let remainder = 10 % 0;

console.log(remainder); *// NaN*

Code language: JavaScript (javascript)

If both dividend and divisor are Infinity, the remainder is NaN:

let remainder = Infinity % Infinity;

console.log(remainder); *// NaN*

Code language: JavaScript (javascript)

If a dividend is a finite number and the divisor is an Infinity, the remainder is the dividend. For example:

let remainder = 10 % Infinity;

console.log(remainder); *// 10*

Code language: JavaScript (javascript)

If the dividend is zero and the divisor is non-zero, the remainder is zero:

let remainder = 0 % 10;

console.log(remainder); *// 0*

Code language: JavaScript (javascript)

If either dividend or divisor is not a number, it’s converted to a number using the Number() function and applied the above rules. For example:

let remainder = '10' % 3;

console.log(remainder); *// 1*

Code language: JavaScript (javascript)

## Using the remainder operator to check if a number is an odd number

To check if a number is an odd number, you use the remainder operator (%) like the following example:

let num = 13;

let isOdd = num % 2;

Code language: JavaScript (javascript)

In this example, if the num is an odd number, the remainder is one. But if the num is an even number, the remainder is zero.

Later, you’ll learn how to [define a function](https://www.javascripttutorial.net/javascript-function/) that returns true if a number is odd or false otherwise like this:

function isOdd(num) {

return num % 2;

}

Code language: JavaScript (javascript)

Or using an [arrow function in ES6](https://www.javascripttutorial.net/es6/javascript-arrow-function/):

const isOdd = (num) => num % 2;

Code language: JavaScript (javascript)

## Remainder vs Modulo operator

In JavaScript, the remainder operator (%) is not the modulo operator.

If you have been working with Python, you may find the % represents the modulo operator in this language. However, it is not the case in JavaScript.

To get a modulo in JavaScript, you use the following expression:

((dividend % divisor) + divisor) % divisor

Or wrap it in a function:

const mod = (dividend, divisor) => ((dividend % divisor) + divisor) % divisor;

Code language: JavaScript (javascript)

If the division and divisor have the same sign, the remainder and modulo operators return the same result. Otherwise, they return different results.

For example:

const mod = (dividend, divisor) => ((dividend % divisor) + divisor) % divisor;

*// dividen and divisor have the same sign*

console.log('remainder:', 5 % 3); *// 2*

console.log('modulo:', mod(5, 3)); *// 2*

*// dividen and divisor have the different signs*

console.log('remainder:', -5 % 3); *// -2*

console.log('modulo:', mod(-5, 3)); *// 1*

Code language: JavaScript (javascript)

Output:

remainder: 2

modulo: 2

remainder: -2

modulo: 1

Code language: HTTP (http)

## Summary

* Use the JavaScript remainder operator (%) get the the remainder of a value divided by another value.

JavaScript Assignment Operators

**Summary**: in this tutorial, you will learn how to use JavaScript assignment operators to assign a value to a variable.

Introduction to JavaScript assignment operators

An assignment operator (=) assigns a value to a variable. The syntax of the assignment operator is as follows:

let a = b;

Code language: JavaScript (javascript)

In this syntax, JavaScript evaluates the expression b first and assigns the result to the variable a.

The following example declares the counter variable and initializes its value to zero:

let counter = 0;

Code language: JavaScript (javascript)

The following example increases the counter variable by one and assigns the result to the counter variable:

let counter = 0;

counter = counter + 1;

Code language: JavaScript (javascript)

When evaluating the second statement, JavaScript evaluates the expression on the right hand first (counter + 1) and assigns the result to the counter variable. After the second assignment, the counter variable is 1.

To make the code more concise, you can use the += operator like this:

let counter = 0;

counter += 1;

Code language: JavaScript (javascript)

In this syntax, you don’t have to repeat the counter variable twice in the assignment.

The following table illustrates assignment operators that are shorthand for another operator and the assignment:

| **Operator** | **Meaning** | **Description** |
| --- | --- | --- |
| a = b | a = b | Assigns the value of b to a. |
| a += b | a = a + b | Assigns the result of a plus b to a. |
| a -= b | a = a - b | Assigns the result of a minus b to a. |
| a \*= b | a = a \* b | Assigns the result of a times b to a. |
| a /= b | a = a / b | Assigns the result of a divided by b to a. |
| a %= b | a = a % b | Assigns the result of a modulo b to a. |
| a &=b | a = a & b | Assigns the result of a AND b to a. |
| a |=b | a = a | b | Assigns the result of a OR b to a. |
| a ^=b | a = a ^ b | Assigns the result of a XOR b to a. |
| a <<= b | a = a << b | Assigns the result of a shifted left by b to a. |
| a >>= b | a = a >> b | Assigns the result of a shifted right (sign preserved) by b to a. |
| a >>>= b | a = a >>> b | Assigns the result of a shifted right by b to a. |

Chaining JavaScript assignment operators

If you want to assign a single value to multiple variables, you can chain the assignment operators. For example:

let a = 10, b = 20, c = 30;

a = b = c; *// all variables are 30*

Code language: JavaScript (javascript)

In this example, JavaScript evaluates from right to left. Therefore, it does the following:

let a = 10, b = 20, c = 30;

b = c; *// b is 30*

a = b; *// a is also 30*

Code language: JavaScript (javascript)

Summary

* Use the assignment operator (=) to assign a value to a variable.
* Chain the assignment operators if you want to assign a single value to multiple variables.

JavaScript Unary Operators Explained

**Summary**: in this tutorial, you will learn how to use JavaScript unary operators that take a single operand and performs an operation.

Introduction to the JavaScript unary operators

Unary operators work on one value. The following table shows the unary operators and their meanings:

| **Unary Operators** | **Name** | **Meaning** |
| --- | --- | --- |
| +x | Unary Plus | Convert a value into a number |
| -x | Unary Minus | Convert a value into a number and negate it |
| ++x | Increment Operator (Prefix) | Add one to the value |
| –x | Decrement Operator (Prefix) | Subtract one from the value |
| x++ | Increment Operator (Postfix) | Add one to the value |
| x– | Decrement Operator (Postfix) | Subtract one from the value |

Unary plus (+)

The unary plus operator is a simple plus sign (+). If you place the unary plus before a numeric value, it does nothing. For example

let x = 10;

let y = +x;

console.log(y); *// 10*

Code language: JavaScript (javascript)

When you apply the unary plus operator to a non-numeric value, it performs a number conversion using the Number() function wit the rules in the following table:

| **Value** | **Result** |
| --- | --- |
| boolean | false to 0, true to 1 |
| string | Convert the string value based on a set of specific rules |
| object | Call the valueOf() and/or toString() method to get the value to convert into a number |

For example, the following uses the unary plus operator to convert the string '10' to the number 10:

let s = '10';

console.log(+s); *// 10*

Code language: JavaScript (javascript)

The following example uses the unary plus operator (+) converts a boolean value into a number, false to 0 and true to 1.

let f = false,

t = true;

console.log(+f); *// 0*

console.log(+t); *// 1*

Code language: JavaScript (javascript)

Output:

0

1

Suppose you have a product object with the toString() method as follows:

let person = {

name: 'John',

toString: function () {

return '25';

},

};

console.log(+person);

Code language: JavaScript (javascript)

Output:

25

In this example, we apply the unary plus operator (+) on the person object that has the toString() method, JavaScript engine calls toString() method to get the value ('25') and convert it into a number. The following adds the valueOf() method to the person object:

let person = {

name: 'John',

toString: function () {

return '25';

},

valueOf: function () {

return '30';

},

};

console.log(+person);

Code language: JavaScript (javascript)

Output:

30

In this example, the person object has the valueOf() method, the JavaScript engine calls it instead of the toString() method to get the value to convert.

Unary minus (-)

The unary minus operator is a single minus sign (-). If you apply the unary minus operator to a number, it negates the number. For example:

let x = 10;

let y = -x;

console.log(y); *// -10*

Code language: JavaScript (javascript)

If you apply the unary minus operator to a non-numeric value, it converts the value into a number using the same rules as the unary plus operator and then negates the value.

Increment / Decrement operators

The increment operator has two plus signs (++) while the decrement operator has two minus signs (--).

Both increment and decrement operators have two versions: prefix and postfix. And you place the prefix and postfix versions of the increment or decrement operators before and after the variable to which they apply.

The following example uses the prefix increment operator to add one to a variable:

let age = 25;

++age;

console.log(age); *// 26*

Code language: JavaScript (javascript)

It’s equivalent to the following:

let age = 25;

age = age + 1;

console.log(age); *// 26*

Code language: JavaScript (javascript)

The following example uses the prefix decrement operator to subtract one from a variable:

let weight = 90;

--weight;

console.log(weight); *// 89*

Code language: JavaScript (javascript)

It is equivalent to the following:

let weight = 90;

weight = weight - 1;

console.log(weight); *// 89*

Code language: JavaScript (javascript)

When you apply the prefix increment or decrement, JavaScript changes the variable before evaluating the statement. For example:

let weight = 90;

weight = ++weight + 5;

console.log(weight); *// 96*

Code language: JavaScript (javascript)

In this example:

* First, increase the weight on the right-hand side so ++weight is 91
* Second, add five to the ++weight that returns 96
* Third, assign the result to the weight on the left-hand side.

Likewise, the following example uses a prefix decrement operator:

let weight = 90;

weight = --weight + 5;

console.log(weight); *// 94*

Code language: JavaScript (javascript)

In this example:

* First, subtract one from the weight, –weight returns 89
* Second, add five to the –weight that returns 94
* Third, assign the result to the weight on the left-hand side.

The postfix increment or decrement operator changes the value after the statement is evaluated. For example:

let weight = 90;

let newWeight = weight++ + 5;

console.log(newWeight); *// 95*

console.log(weight); *// 91*

Code language: JavaScript (javascript)

How it works.

* First, add five to weight (90) and assign the result to the newWeight (95)
* Second, add one to the weight variable after the second statement completes, the weight becomes 91.
* Third, output both newWeight and weight to the console.

When applying the increment/decrement operator to a non-numeric value, it performs the following steps :

* First, convert the value into a number using the same rules as the unary plus (+) operator.
* Then, add one to or subtract one from the value.

Summary

* Unary operators work on one value.
* Unary plus (+) or minus (-) converts a non-numeric value into a number. The unary minus negates the value after the conversion.
* The prefix increment operator adds one to a value. The value is changed before the statement is evaluated.
* The postfix increment operator adds one to a value. The value is changed after the statement is evaluated.
* The prefix decrement operator subtracts one from a value. The value is changed before the statement is evaluated.
* The postfix decrement operator subtracts one from a value. The value is changed after the statement is evaluated.

# JavaScript Comparison Operators

**Summary**: in this tutorial, you will learn how to use JavaScript comparison operators to compare two values.

## Introduction to JavaScript comparison operators

To compare two values, you use a comparison operator. The following table shows the comparison operators in JavaScript:

| **Operator** | **Meaning** |
| --- | --- |
| < | less than |
| > | greater than |
| <= | less than or equal to |
| >= | greater than or equal to |
| == | equal to |
| != | not equal to |

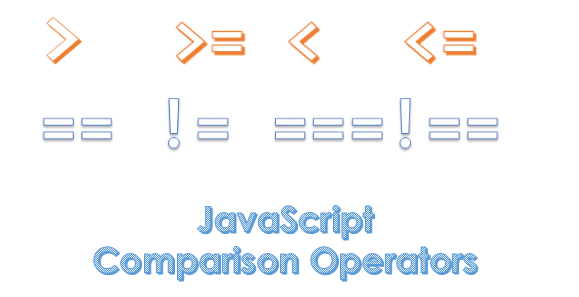
A comparison operator returns a [Boolean](https://www.javascripttutorial.net/javascript-boolean/) value indicating that the comparison is true or not. See the following example:

let r1 = 20 > 10; *// true*

let r2 = 20 < 10; *// false*

let r3 = 10 == 10; *// true*

Code language: JavaScript (javascript)



A comparison operator takes two values. If the types of the values are not comparable, the comparison operator converts them into values of comparable types according to specific rules.

### **Compare numbers**

If values are numbers, the comparison operators perform a numeric comparison. For example:

let a = 10,

b = 20;

console.log(a >= b); *// false*

console.log(a == 10); *// true*

Code language: JavaScript (javascript)

This example is straightforward. The variable a is 10, b is 20. The expression a >= b expression returns false and the expression a == 10 expression returns true.

### **Compare strings**

If the operands are strings, JavaScript compares the character codes numerically one by one in the string.

let name1 = 'alice',

name2 = 'bob';

let result = name1 < name2;

console.log(result); *// true*

console.log(name1 == 'alice'); *// true*

Code language: JavaScript (javascript)

Because JavaScript compares the character codes in the strings numerically, you may receive an unexpected result, for example:

let f1 = 'apple',

f2 = 'Banana';

let result = f2 < f1;

console.log(result); *// true*

Code language: JavaScript (javascript)

In this example, f2 is less than f1 because the letter B has the character code 66 while the letter a has the character code 97.

To fix this, you need to:

* First, convert the strings into a common format, either lowercase or uppercase
* Second, compare the converted values

For example:

let f1 = 'apple',

f2 = 'Banana';

let result = f2.toLowerCase() < f1.toLowerCase();

console.log(result); *// false*

Code language: JavaScript (javascript)

Note that the toLowerCase() is a method of the String object that converts the string to lowercase.

### **Compare a number with a value of another type**

If a value is a number and the other is not, the comparison operator will convert the non-numeric value into a number and compare them numerically. For example:

console.log(10 < '20'); *// true*

Code language: JavaScript (javascript)

In this example, the comparison operator converts the string '20' into the number 20 and compares with the number 10. Here is an example:

console.log(10 == '10'); *// true*

Code language: JavaScript (javascript)

In this example, the comparison operator converts the string '10' into the number 10 and compares them numerically.

### **Compare an object with a non-object**

If a value is an object, the valueOf() method of that object is called to return the value for comparison. If the object doesn’t have the valueOf() method, the toString() method is called instead. For example:

let apple = {

valueOf: function () {

return 10;

},

};

let orange = {

toString: function () {

return '20';

},

};

console.log(apple > 10); *// false*

console.log(orange == 20); *// true*

Code language: JavaScript (javascript)

In this first comparison, the apple object has the valueOf() method that returns 10. Therefore, the comparison operator uses the number 10 for comparison.

In the second comparison, JavaScript first calls the valueOf() method. However, the orange object doesn’t have the valueOf() method. So JavaScript calls the toString() method to get the returned value of 20 for comparison.

### **Compare a Boolean with another value**

If a value is a Boolean value, JavaScript converts it to a number and compares the converted value with the other value; true is converted to 1 and false is converted to 0. For example:

console.log(true > 0); *// true*

console.log(false < 1); *// true*

console.log(true > false); *// true*

console.log(false > true); *// false*

console.log(true >= true); *// true*

console.log(true <= true); *// true*

console.log(false <= false); *// true*

console.log(false >= false); *// true*

Code language: JavaScript (javascript)

In addition to the above rules, the equal (==) and not-equal(!=) operators also have the following rules.

### **Compare null and undefined**

In JavaScript, null equals undefined. It means that the following expression returns true.

console.log(null == undefined); *// true*

Code language: JavaScript (javascript)

### **Compare NaN with other values**

If either value is NaN, then the equal operator(==) returns false.

console.log(NaN == 1); *// false*

Code language: JavaScript (javascript)

Even

console.log(NaN == NaN); *// false*

Code language: JavaScript (javascript)

The not-equal (!=) operator returns true when comparing the NaN with another value:

console.log(NaN != 1); *// true*

Code language: JavaScript (javascript)

And also

console.log(NaN != NaN); *// true*

Code language: JavaScript (javascript)

## Strict equal (===) and not strict equal (!==)

Besides the comparison operators above, JavaScript provides the strict equal ( ===) and not strict equal  ( !==) operators.

| **Operator** | **Meaning** |
| --- | --- |
| === | strict equal |
| !== | not strict equal |

The strict equal and not strict equal operators behave like the equal and not equal operators except that they don’t convert the operand before comparison. See the following example:

console.log("10" == 10); *// true*

console.log("10" === 10); *// false*

Code language: JavaScript (javascript)

In the first comparison, since we use the equality operator, JavaScript converts the string into the number and performs the comparison.

However, in the second comparison, we use the strict equal operator ( ===), JavaScript doesn’t convert the string before comparison, therefore the result is false.

In this tutorial, you have learned how to use the JavaScript comparison operators to compare values.

# An Introduction to JavaScript Logical Operators



**Summary**: in this tutorial, you will learn how to use the JavaScript logical operators including the logical NOT operator( !), the logical AND operator ( &&) and the logical OR operator ( ||).

The logical operators are important in JavaScript because they allow you to compare [variables](https://www.javascripttutorial.net/javascript-variables/) and do something based on the result of that comparison.

For example, if the result of the comparison is true, you can run a block of code; if it’s false, you can execute another code block.

JavaScript provides three logical operators:

* ! (Logical NOT)
* || (Logical OR)
* && (Logical AND)

## 1) The Logical NOT operator (!)

JavaScript uses an exclamation point ! to represent the logical NOT operator. The ! operator can be applied to a single value of any type, not just a Boolean value.

When you apply the ! operator to a boolean value, the ! returns true if the value is false and vice versa. For example:

let eligible = false,

required = true;

console.log(!eligible);

console.log(!required);

Code language: JavaScript (javascript)

Output:

true

false

Code language: JavaScript (javascript)

In this example, the eligible is true so !eligible returns false. And since the required is true, the !required returns false.

When you apply the ! operator to a non-Boolean value. The ! operator first converts the value to a [boolean](https://www.javascripttutorial.net/javascript-data-types/" \l "boolean) value and then negates it.

The following example shows how to use the ! operator:

!a

The logical ! operator works based on the following rules:

* If a is [undefined](https://www.javascripttutorial.net/javascript-undefined/), the result is true.
* If a is [null](https://www.javascripttutorial.net/object/javascript-null/), the result is true.
* If a is a number other than 0, the result is false.
* If a is [NaN](https://www.javascripttutorial.net/javascript-nan/), the result is true.
* If a is an object, the result is false.
* If a is an empty string, the result is true. In the case a is a non-empty string, the result is false

The following demonstrates the results of the logical ! operator when applying to a non-boolean value:

console.log(!undefined); *// true*

console.log(!null); *// true*

console.log(!20); *//false*

console.log(!0); *//true*

console.log(!NaN); *//true*

console.log(!{}); *// false*

console.log(!''); *//true*

console.log(!'OK'); *//false*

console.log(!false); *//true*

console.log(!true); *//false*

Code language: JavaScript (javascript)

### **Double-negation (!!)**

Sometimes, you may see the double negation (!!) in the code. The !! uses the logical NOT operator (!) twice to convert a value to its real boolean value.

The result is the same as using the [Boolean()](https://www.javascripttutorial.net/javascript-boolean/) function. For example:

let counter = 10;

console.log(!!counter); *// true*

Code language: JavaScript (javascript)

The first ! operator negates the Boolean value of the counter variable. If the counter is true, then the ! operator makes it false and vice versa.

The second ! operator negates that result of the first ! operator and returns the real boolean value of the counter variable.

## 2) The Logical AND operator (&&)

JavaScript uses the double ampersand (&&) to represent the logical AND operator. The following expression uses the && operator:

let result = a && b;

Code language: JavaScript (javascript)

If a can be converted to true, the && operator returns the b; otherwise, it returns the a. In fact, this rule is applied to all boolean values.

The following truth table illustrates the result of the && operator when it is applied to two Boolean values:

| **a** | **b** | **a && b** |
| --- | --- | --- |
| true | true | true |
| true | false | false |
| false | true | false |
| false | false | false |

The result of the && operator is true only if both values are true, otherwise, it is false. For example:

let eligible = false,

required = true;

console.log(eligible && required); *// false*

Code language: JavaScript (javascript)

In this example, the eligible is false, therefore, the value of the expression eligible && required is false.

See the following example:

let eligible = true,

required = true;

console.log(eligible && required); *// true*

Code language: JavaScript (javascript)

In this example, both eligible and required are true, therefore, the value of the expression eligible && required is true.

### **Short-circuit evaluation**

The && operator is short-circuited. It means that the && operator evaluates the second value only if the first one doesn’t suffice to determine the value of an expression. For example:

let b = true;

let result = b && (1 / 0);

console.log(result);

Code language: JavaScript (javascript)

Output:

Infinity

Code language: JavaScript (javascript)

In this example, b is true therefore the && operator could not determine the result without further evaluating the second expression (1/0).

The result is Infinity which is the result of the expression (1/0). However:

let b = false;

let result = b && (1 / 0);

console.log(result);

Code language: JavaScript (javascript)

Output:

false

Code language: JavaScript (javascript)

In this case, b is false, the && operator doesn’t need to evaluate the second expression because it can determine the final result as false based value of the first value.

### **The chain of && operators**

The following expression uses multiple && operators:

let result = value1 && value2 && value3;

Code language: JavaScript (javascript)

The && operator carries the following:

* Evaluates values from left to right.
* For each value, converts it to a boolean. If the result is false, stops and returns the original value.
* If all values are truthy values, returns the last value.

In other words, The && operator returns the first falsy value or the last value if none were found.

If a value can be converted to true, it is so-called a truthy value. If a value can be converted to false, it is a so-called falsy value.

## 3) The Logical OR operator (||)

JavaScript uses the double pipe || to represent the logical OR operator. You can apply the || operator to two values of any type:

let result = a || b;

Code language: JavaScript (javascript)

If a can be converted to true, returns a; else, returns b. This rule is also applied to boolean values.

The following truth table illustrates the result of the || operator based on the value of the operands:

| **a** | **b** | **a || b** |
| --- | --- | --- |
| true | true | true |
| true | false | true |
| false | true | true |
| false | false | false |

The || operator returns false if both values evaluate to false. In case either value is true, the || operator returns true. For example:

let eligible = true,

required = false;

console.log(eligible || required); *// true*

Code language: JavaScript (javascript)

See another example:

let eligible = false,

required = false;

console.log(eligible || required); *// false*

Code language: JavaScript (javascript)

In this example, the expression eligible || required returns false because both values are false.

### **The || operator is also short-circuited**

Similar to the && operator, the || operator is short-circuited. It means that if the first value evaluates to true, the && operator doesn’t evaluate the second one.

### **The chain of || operators**

The following example shows how to use multiple || operators in an expression:

let result = value1 || value2 || value3;

Code language: JavaScript (javascript)

The || operator does the following:

* Evaluates values from left to right.
* For each value, converts it to a boolean value. If the result of the conversion is true, stops and returns the value.
* If all values have been evaluated to false, returns the last value.

In other words, the chain of the || operators returns the first truthy value or the last one if no truthy value was found.

## Logical operator precedence

When you mix logical operators in an expression, the JavaScript engine evaluates the operators based on a specified order. And this order is called the operator precedence.

In other words, the operator precedence is the order of evaluation of logical operators in an expression.

The precedence of the logical operator is in the following order from the highest to the lowest:

1. Logical NOT (!)
2. Logical AND (&&)
3. Logical OR (||)

## Summary

* The NOT operator (!) negates a boolean value. The (!!) converts a value into its real boolean value.
* The AND operator (&&) is applied to two Boolean values and returns true if both values are true.
* The OR operator (||) is applied to two Boolean values and returns true if one of the operands is true.
* Both && and || operator are short-circuited. They can be also applied to non-Boolean values.
* The logical operator precedence from the highest to the lowest is !, && and ||.

JavaScript Logical Assignment Operators

**Summary**: in this tutorial, you’ll learn about JavaScript logical assignment operators, including the logical OR assignment operator (||=), the logical AND assignment operator (&&=), and the nullish assignment operator (??=).

ES2021 introduces three logical assignment operators including:

* Logical OR assignment operator (||=)
* Logical AND assignment operator (&&=)
* Nullish coalescing assignment operator (??=)

The following table shows the equivalent of the logical assignments operator:

| **Logical Assignment Operators** | **Logical Operators** |
| --- | --- |
| x ||= y | x || (x = y) |
| x &&= y | x && (x = y) |
| x ??= y | x ?? (x = y); |

The Logical OR assignment operator

The logical OR assignment operator (||=) accepts two operands and assigns the right operand to the left operand if the left operand is falsy:

x ||= y

In this syntax, the ||= operator only assigns y to x if x is falsy. For example:

let title;

title ||= 'untitled';

console.log(title);

Code language: JavaScript (javascript)

Output:

untitled

In this example, the title variable is undefined, therefore, it’s falsy. Since the title is falsy, the operator ||= assigns the 'untitled' to the title. The output shows the untitled as expected.

See another example:

let title = 'JavaScript Awesome';

title ||= 'untitled';

console.log(title);

Code language: JavaScript (javascript)

Output:

'JavaScript Awesome'

Code language: JavaScript (javascript)

In this example, the title is 'JavaScript Awesome' so it is truthy. Therefore, the logical OR assignment operator (||=) doesn’t assign the string 'untitled' to the title variable.

The logical OR assignment operator:

x ||= y

is equivalent to the following statement that uses the [logical OR operator](https://www.javascripttutorial.net/javascript-logical-operators/):

x || (x = y)

Like the logical OR operator, the logical OR assignment also short-circuits. It means that the logical OR assignment operator only performs an assignment when the x is falsy.

The following example uses the logical assignment operator to display a default message if the search result element is empty:

document.querySelector('.search-result').textContent ||= 'Sorry! No result found';

Code language: JavaScript (javascript)

The Logical AND assignment operator

The logical AND assignment operator only assigns y to x if x is truthy:

x &&= y;

The logical AND assignment operator also short-circuits. It means that

x &&= y;

is equivalent to:

x && (x = y);

The following example uses the logical AND assignment operator to change the last name of a person object if the last name is truthy:

let person = {

firstName: 'Jane',

lastName: 'Doe',

};

person.lastName &&= 'Smith';

console.log(person);

Code language: JavaScript (javascript)

Output:

{firstName: 'Jane', lastName: 'Smith'}

Code language: CSS (css)

The nullish coalescing assignment operator

The nullish coalescing assignment operator only assigns y to x if x is null or undefined:

x ??= y;

It’s equivalent to the following statement that uses the [nullish coalescing operator](http://javascript%20nullish%20coalescing%20operahttps//www.javascripttutorial.net/es-next/javascript-nullish-coalescing-operator/tor):

x ?? (x = y);

The following example uses the nullish coalescing assignment operator to add a missing property to an object:

let user = {

username: 'Satoshi'

};

user.nickname ??= 'anonymous';

console.log(user);

Code language: JavaScript (javascript)

Output:

{username: 'Satoshi', nickname:'anonymous'}

Code language: CSS (css)

In this example, the user.nickname is undefined, therefore, it’s nullish. The nullish coalescing assignment operator assigns the string 'anonymous' to the user.nickname property.

The following table illustrates how the logical assignment operators work:

Summary

* The logical OR assignment (x ||= y) operator only assigns y to x if x is falsy.
* The logical AND assignment (x &&= y) operator only assigns y to x if x is truthy.
* The nullish coalescing assignment (x ??= y) operator only assigns y to x if x is nullish.

JavaScript Nullish Coalescing Operator

**Summary**: in this tutorial, you’ll learn about the JavaScript nullish coalescing operator (??) that accepts two values and returns the second value if the first one is null or undefined.

Introduction to the JavaScript nullish coalescing operator

ES2020 introduced the nullish coalescing operator denoted by the double question marks (??). The nullish coalescing operator is a [logical operator](https://www.javascripttutorial.net/javascript-logical-operators/) that accepts two values:

value1 ?? value2

The nullish coalescing operator returns the second value (value2) if the first value (value2) is null or undefined. Technically, the nullish coalescing operator is equivalent to the following block:

const result = value1;

if(result === null || result === undefined) {

result = value2;

}

Code language: JavaScript (javascript)

A nullish value is a value that is either null or undefined.

The following example uses the nullish coalescing operator (??) to return the string 'John' because the first value is null:

const name = null ?? 'John';

console.log(name); *// 'John'*

Code language: JavaScript (javascript)

And this example returns 28 because the first value is undefined:

const age = undefined ?? 28;

console.log(age);

Code language: JavaScript (javascript)

Why nullish coalescing operator

When assigning a default value to a [variable](https://www.javascripttutorial.net/javascript-variables/), you often use the [logical OR operator](https://www.javascripttutorial.net/javascript-logical-operators/) (||). For example:

let count;

let result = count || 1 ;

console.log(result); *// 1*

Code language: JavaScript (javascript)

In this example, the count variable is undefined, it is coerced to false. Therefore, the result is 1.

However, the logical OR operator (||) sometimes is confusing if you consider 0 or empty strings '' as a valid value like this:

let count = 0;

let result = count || 1;

Code language: JavaScript (javascript)

The result is 1, not 0, which you may not expect.

The nullish coalescing operator helps you to avoid this pitfall. It only returns the second value when the first one is either null or undefined.

The nullish coalescing operator is short-circuited

Similar to the [logical OR and AND operators](https://www.javascripttutorial.net/javascript-logical-operators/), the nullish coalescing operator does not evaluate the second value if the first operand is neither undefined nor null.

For example:

let result = 1 ?? console.log('Hi');

Code language: JavaScript (javascript)

In this example, the operator ?? does not evaluate the second expression that displays the “Hi” to the console because the first value is 1, which is not null or undefined.

The following example evaluates the second expression because the first one is undefined:

let result = undefined ?? console.log('Hi');

Code language: JavaScript (javascript)

Output:

'Hi'

Code language: JavaScript (javascript)

Chaining with the AND or OR operator

A SyntaxError will occur if you combine the logical AND or OR operator directly with the nullish coalescing operator like this:

const result = null || undefined ?? 'OK'; *// SyntaxError*

Code language: JavaScript (javascript)

However, you can avoid this error by wrapping the expression on the left of the ?? operator in parentheses to explicitly specify the operator precedences:

const result = (null || undefined) ?? 'OK';

console.log(result); *// 'OK'*

Code language: JavaScript (javascript)

Summary

* The nullish coalescing operator (??) is a logical operator that accepts two values and returns the second value if the first one is null or undefined.
* The nullish coalescing operator is short-circuited and cannot directly combine with the logical AND or OR operator.

JavaScript Exponentiation Operator

**Summary**: in this tutorial, you will learn how to use the JavaScript exponentiation operator (\*\*) to raise a number to the power of an exponent.

Introduction to the JavaScript exponentiation operator

To raise a number to the power of an exponent, you often use the static method Math.pow() with the following syntax:

Math.pow(base, exponent)

Code language: JavaScript (javascript)

For example:

let result = Math.pow(2,2);

console.log(result); *// 4*

result = Math.pow(2,3);

console.log(result); *// 8*

Code language: JavaScript (javascript)

ECMAScript 2016 provided an alternative way to get a base to the exponent power by using the exponentiation operator ( \*\*) with the following syntax:

x\*\*n

The operator \*\* raises the x to the power of an exponent n.

Note that some languages use the caret symbol ^ for exponentiation. However, JavaScript already uses that symbol for the bitwise logical XOR operator.

The following example illustrates how to use the exponentiation operator (\*\*):

let result = 2 \*\* 2;

console.log(result); *// 4*

result = 2 \*\* 3;

console.log(result); *// 8*

Code language: JavaScript (javascript)

The Math.pow() accepts a value and converts it to a value of the number type for calculation. Similarly, the operator \*\* accept values of the number type. In addition, the operator \*\* accepts a value of the [bigint](https://www.javascripttutorial.net/es-next/javascript-bigint/) type. For example:

let result = 2n \*\* 3n;

console.log(result); *// 8n*

Code language: JavaScript (javascript)

Also, you can use the exponentiation operator  ( \*\*) in the infix notation. For example:

let x = 2;

x \*\*= 4;

console.log(x); *// 16*

Code language: JavaScript (javascript)

JavaScript does not allow you to put a unary operator immediately before the base number. If you attempt to do so, you’ll get a SyntaxError.

The following example causes a syntax error:

let result = -2\*\*3;

Code language: JavaScript (javascript)

Error:

Uncaught SyntaxError: Unary operator used immediately before exponentiation expression. Parenthesis must be used to disambiguate operator precedence

Code language: JavaScript (javascript)

To fix this, you use parentheses like this:

let result = (-2)\*\*3;

console.log(result); *// -8*

Code language: JavaScript (javascript)

Summary

* The exponentiation operator \*\* raises a number to the power of an exponent.
* The exponentiation operator \*\* accepts values of the type number or bigint.