https://digiimento.com

Have any question ? + 91 9821876104 / 06 admin@gatelectures.com +91 9821876104 **LAST DAY: Special Discount on 1 Year & 6 Months course.** CLICK HERE

gatelectures.com

Home Courses About Us Videos Classroom Program Downloads Contact Login

#ThinkBig

Trust of 1 Lakh + Students.

Celebrating more than 1 Lakh Followers on Social Media .

Ratings : 4.8

VIDEOS

Operators in JavaScript

Table 1 operator precedence and associativity in JavaScript			
Operator	Operator Use	Operator Associativity	Operator Precedence
()	Method/function call, grouping	Left to right	Highest — 1
[]	Array access	Left to right	1
.	Object property access	Left to right	1

++	Increment	Right to left	2
--	Decrement	Right to left	2
-	Negation	Right to left	2
!	Logical NOT	Right to left	2
~	Bitwise NOT	Right to left	2
delete	Removes array value or object property	Right to left	2
new	Creates an object	Right to left	2
typeof	Returns data type	Right to left	2
void	Specifies no value to return	Right to left	2

/	Division	Left to right	3
*	Multiplication	Left to right	3
%	Modulus	Left to right	3
+	Plus	Left to right	4
+	String Concatenation	Left to right	4
-	Subtraction	Left to right	4
>>	Bitwise right-shift	Left to right	5
<<	Bitwise left-shift	Left to right	5
>, >=	Greater than, greater than or equal to	Left to right	6
<, <=	Less than, less than or equal to	Left to right	6

==	Equality	Left to right	7
!=	Inequality	Left to right	7
===	Identity operator — equal to (and same data type)	Left to right	7
!==	Non-identity operator — not equal to (or don't have the same data type)	Left to right	7
&	Bitwise AND	Left to right	8
^	Bitwise XOR	Left to right	9
|	Bitwise OR	Left to right	10
&&	Logical AND	Left to right	11
||	Logical OR	Left to right	12
?:	Conditional branch	Left to right	13
=	Assignment	Right to left	14
*=, /=, %=, +=,, -=, <<=, >>=, >>>=, &=, ^=, |=	Assignment according to the preceding operator	Right to left	14
,	Multiple evaluation	Left to right	Lowest: 15

Strings concatenation, binary +

- Usually the plus operator + sums numbers.
- But if the binary + is applied to strings, it merges (concatenates) them

```
<body>
  <script>
    'use strict';
    alert( '1' + 2 ); // "12"
    alert( 2 + '1' ); // "21"
    alert(2 + 2 + '1' ); // "41" and not "221"
    alert( 2 - '1' ); // 1
    alert( '6' / '2' ); // 3
    // No effect on numbers
    let x = 1;
    alert( +x ); // 1
    let y = -2;
    alert( +y ); // -2
    // Converts non-numbers
    alert( +true ); // 1
    alert( +"" );   // 0
  </script>
</body>
```

The plus + exists in two forms. The binary form that we used above and the unary form.
The unary plus or, in other words, the plus operator + applied to a single value, doesn't do anything with numbers, but if the operand is not a number, then it is converted into it.
It actually does the same as Number(...), but is shorter

```
<body>
  <script>
    'use strict';
    let apples = "2";
    let oranges = "3";
    alert( apples + oranges );
    // "23", the binary plus concatenates strings
    // both values converted to numbers before the binary plus
    alert( +apples + +oranges ); // 5
    // the longer variant
    // alert( Number(apples) + Number(oranges) ); // 5
  </script>
</body>
```

The assignment operator `"="` returns a value

An operator always returns a value. That's obvious for most of them like an addition `+` or a multiplication `*`. But the assignment operator follows that rule too.

The call `x = value` writes the `value` into `x` *and then returns it*.

Here's the demo that uses an assignment as part of a more complex expression:

```
1 let a = 1;
2 let b = 2;
3
4 let c = 3 - (a = b + 1);
5
6 alert( a ); // 3
7 alert( c ); // 0
```

In the example above, the result of `(a = b + 1)` is the value which is assigned to `a` (that is `3`). It is then used to subtract from `3`.

Funny code, isn't it? We should understand how it works, because sometimes we can see it in 3rd-party libraries, but shouldn't write anything like that ourselves. Such tricks definitely don't make the code clearer and readable.

Exponentiation **

The exponentiation operator `**` is a recent addition to the language.

For a natural number `b`, the result of `a ** b` is `a` multiplied by itself `b` times.

For instance:

```
alert( 2 ** 2 ); // 4  (2 * 2)
alert( 2 ** 3 ); // 8  (2 * 2 * 2)
alert( 2 ** 4 ); // 16 (2 * 2 * 2 * 2)
```

The operator works for non-integer numbers of `a` and `b` as well, for instance:

```
alert( 4 ** (1/2) ); // 2 (power of 1/2 is the same as a square root,
alert( 8 ** (1/3) ); // 2 (power of 1/3 is the same as a cubic root)
```

Comma

The comma operator `,` is one of most rare and unusual operators. Sometimes it's used to write shorter code, so we need to know it in order to understand what's going on.

The comma operator allows us to evaluate several expressions, dividing them with a comma `,`. Each of them is evaluated, but the result of only the last one is returned.

For example:

```
let a = (1 + 2, 3 + 4);

alert( a ); // 7 (the result of 3 + 4)
```

Here, the first expression `1 + 2` is evaluated, and its result is thrown away, then `3 + 4` is evaluated and returned as the result.