Technologies

References & Guides

Feedback

rajeshchauhan23102008

Search

Operator precedence

Operator precedence determines the way in which operators are parsed with respect to each other. Operators with higher precedence become the operands of operators with lower precedence.

JavaScript Demo: Expressions - Operator precedence

```
console.log(3 + 4 * 5); // 3 + 20
// expected output: 23

console.log(4 * 3 ** 2); // 4 * 9
// expected output: 36

var a;
var b;

console.log(a = b = 5);
// expected output: 5;
```

Run ›

Reset

Associativity

Associativity determines the way in which operators of the same precedence are parsed. For example, consider an expression:

```
a OP b OP c
```

Left-associativity (left-to-right) means that it is processed as `(a OP b) OP c`, while right-associativity (right-to-left) means it is interpreted as `a OP (b OP c)`. Assignment operators are right-associative, so you can write:

```
a = b = 5;
```

with the expected result that `a` and `b` get the value 5. This is because the assignment operator returns the value that is assigned. First, `b` is set to 5. Then the `a` is also set to 5, the return value of `b = 5`, aka right operand of the assignment.

Examples

```
3 > 2 && 2 > 1
// returns true

3 > 2 > 1
// returns false because 3 > 2 is true, and true > 1 is false
// Adding parentheses makes things clear: (3 > 2) > 1
```

Table

The following table is ordered from highest (20) to lowest (1) precedence.

Precedence	Operator type	Associativity	Individual operators

20	Grouping	n/a	`( … )`
19	Member Access	left-to-right	`… . …`
	Computed Member Access	left-to-right	`… [ … ]`
	`new` (with argument list)	n/a	`new … ( … )`
	Function Call	left-to-right	`… ( … )`
18	`new` (without argument list)	right-to-left	`new …`
17	Postfix Increment	n/a	`… ++`
	Postfix Decrement		`… --`
16	Logical NOT	right-to-left	`! …`
	Bitwise NOT		`~ …`
	Unary Plus		`+ …`
	Unary Negation		`- …`
	Prefix Increment		`++ …`
	Prefix Decrement		`-- …`
	`typeof`		`typeof …`
	`void`		`void …`
	`delete`		`delete …`
	`await`		`await …`
15	Exponentiation	right-to-left	`… ** …`
14	Multiplication	left-to-right	`… * …`
	Division		`… / …`
	Remainder		`… % …`
13	Addition	left-to-right	`… + …`
	Subtraction		`… - …`
12	Bitwise Left Shift	left-to-right	`… << …`
	Bitwise Right Shift		`… >> …`
	Bitwise Unsigned Right Shift		`… >>> …`

11	Less Than	left-to-right	… < …
	Less Than Or Equal		… <= …
	Greater Than		… > …
	Greater Than Or Equal		… >= …
	`in`		… in …
	`instanceof`		… instanceof …
10	Equality	left-to-right	… == …
	Inequality		… != …
	Strict Equality		… === …
	Strict Inequality		… !== …
9	Bitwise AND	left-to-right	… & …
8	Bitwise XOR	left-to-right	… ^ …
7	Bitwise OR	left-to-right	… | …
6	Logical AND	left-to-right	… && …
5	Logical OR	left-to-right	… || …
4	Conditional	right-to-left	… ? … : …
3	Assignment	right-to-left	… = …
			… += …
			… -= …
			… **= …
			… *= …
			… /= …
			… %= …
			… <<= …
			… >>= …
			… >>>= …
			… &= …

			… `^=` …
			… `|=` …
2	`yield`	right-to-left	`yield` …
	`yield*`		`yield*` …
1	Comma / Sequence	left-to-right	… `,` …