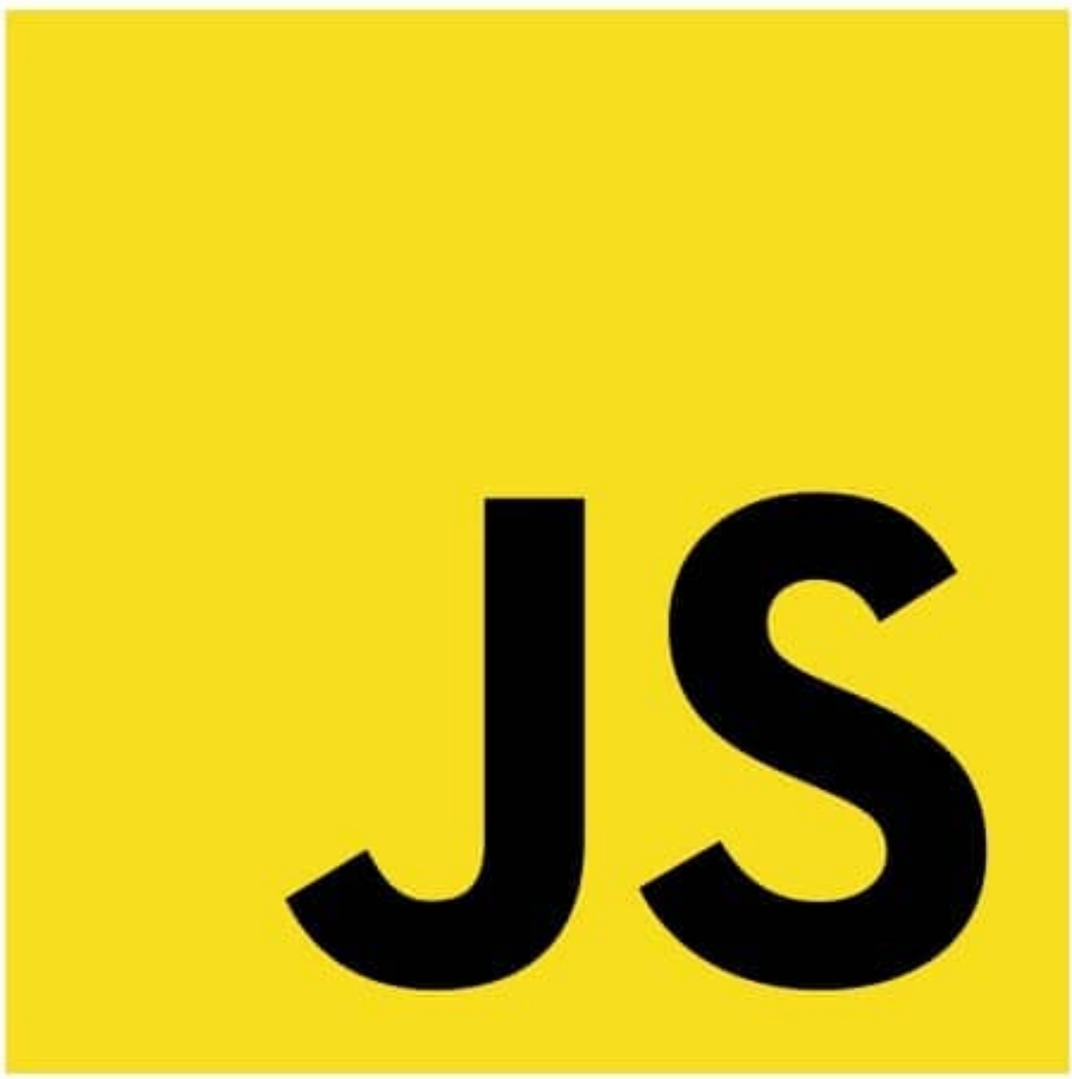
@lupitacode

JAVASCRIPT OPERATORS REFERENCE



/lupitacode

@lupitacode

ARITHMETIC

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

operator	name	example	value
+	Addition	1 + 1;	2
-	Subtraction	5 - 3;	2
*	Multiplication	2 * 2;	4
/	Division	10 / 5;	2
%	Modulus	10 % 4;	2
++	Increment	a++;	a + 1
--	Decrement	a--;	a - 1
**	Exponentiation	2 ** 3;	8

/lupitacode

@lupitacode

ASSIGNMENT

Assignment operators are used to assign values to JavaScript variables

operator	name	example	same as
=	Assignment	a = 1	a = b
+=	Addition	a += b	a = a + b
-=	Subtraction	a -= b	a = a - b
*=	Multiplication	a *= b	a = a * b
/=	Division	a /= b	a = a / b
%=	Modulus	a %= b	a = a % b
**=	Exponentiation	a **= b	a = a ** b
&=	Bitwise AND	a &= b	a = a & b
⊨	Bitwise OR	a ⊨ b	a = a | b
^=	Bitwise XOR	a ^= b	a = a ^ b
≫=	Right shift	a ≫= b	a = a >> b
≪=	Left shift	a ≪= b	a = a << b
>>>=	Unsigned right shift	a >>>= b	a = a >>> b
&&=	Logical AND	a &&= b	a = a &&(a=b)
|⊨	Logical OR	a |⊨ b	a = a ||(a=b)
??=	Logical Nullish	a ??= b	a = a ??(a=b)

@lupitacode

STRING

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

operator	name	example	result
=	Assignment	x = 'a'	'a'
+=	Concatenation	x += 'b'	'ab'
+	Addition	'x' + 'y'	'xy'

LOGICAL

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

operator	name	example	result
&&	Logical AND	(5 > 1 && 3 > 2)	true
||	Logical OR	1 == 1 || 2 == 2	true
!	Logical NOT	!true	false
		!(1 == 2)	true

/lupitacode

@lupitacode

COMPARISON

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

operator	name	example	result
==	equality	1 == 1	true
		'1'== 1	true
		1 == 2	false
===	equality of value *and* type	'1' === '1'	true
		1 === 1	true
		1 === '1'	false
!=	inequality	1 != 1	false
		1 != 2	true
!==	inequality of value *and* type	'1'!== 1	true
		1 !== 1	false
>	greater than	2 > 1	true
<	less than	5 < 7	true
>=	greater or equal	2 >= 1	true
<=	less or equal	2 <= 1	false

* Triple equality operator checks for **value** and **type**.

/lupitacode

@lupitacode

BITWISE

In binary number system decimal numbers have an equivalent represented by a series of 0's and 1's. For example 5 is 0101 and 1 is 0001. Bitwise operators work on those bits, rather than number's decimal values.

operator	name	example	same as	result	decimal
&	AND	x = 5 & 1	0101 & 0001	0001	1
|	OR	x = 5 | 1	0101 | 0001	0101	5
~	NOT	x = ~5	~0101	1010	10
^	XOR	x = 5 ^ 1	0101 ^ 0001	0100	4
<<	Left shift	x = 5 << 1	0101 << 1	1010	10
>>	right shift	x = 5 >> 1	0101 >> 1	0010	2
>>>	Zero-fill right shift	x = 5 >>> 1	0101 >>> 1	0010	2

The << operator is the same as multiplying a whole number by 2 and >> operator is the same as dividing a whole number by 2. They are sometimes used as performance optimizations because they are faster than * and / operators in terms of processor cycles.

@lupitacode

TYPEOF

The typeof operator is used to check the type of a value.
It will often evaluate to either primitive type, object or function.
The value produced by the typeof operator is always string format:

operator	result
`typeof 125;`	`'number'`
`typeof 100n;`	`'bigint'`
`typeof 'text';`	`'string'`
`typeof NaN;`	`'number'`
`typeof true;`	`'boolean'`
`typeof [];`	`'object'`
`typeof {};`	`'object'`
`typeof Object;`	`'function'`
`typeof new Object();`	`'object'`
`typeof null;`	`'object'`

NaN (Not a Number) evaluates to 'number'. This is just one of many JavaScript quirks.

NaN lives natively on **Number.NaN** – it is considered to be a primitive value.

NaN is the symbol usually produced in the context of a numeric operation.

@lupitacode

TERNARY (?:)

The ternary operator is like an inline if-statement.
It does not support {} brackets
or multiple statements.

```
let result = statement ? value : value ;
```

(if: statement; else: value)

DELETE

The delete keyword can be used to delete an object property:

```
const lupita = {
  twitter: "@lupitacode",
  youtube: "youtube.com/lupitacode",
};

console.log(lupita);
//{ twitter: '@lupitacode', youtube: 'youtube.com/lupitacode' }

delete lupita.twitter;
console.log(lupita); //{ youtube: 'youtube.com/lupitacode' }
```

/lupitacode

@lupitacode

IN

The in operator can be used to check if a property name exists in an object.

The in operator, when used together with arrays, will check if an index exists. Note, it is ignorant of actual value (in either arrays or objects.)

```
let trees = ['redwood', 'bay', 'cedar'];
0 in trees;          // returns true
2 in trees;          // returns true
6 in trees;          // returns false
'bay' in trees;      // returns false (you must specify the index number,
                     // not the value at that index)
'length' in trees; // returns true (length is an Array property)
```

- You can check for properties on built-in data types. The length property is native to all arrays

```
'length' in []; //true
'length' in [0, 1, 2]; //true
```

- The "length" property does not exist natively on an object unless it's added explicitly:

```
'length' in {}; //false
'length' in {'length': 1}; //true
```

- Check for presence of constructor or prototype property on an object constructor function

```
"constructor" in Object; //true
"prototype" in Object; //true
```

@lupitacode

INSTANCEOF

Returns true if the specified object is an instance of the specified object

```
let cars = ['Volvo', 'BMW']
cars instanceof Array; //true
```

GROUPING

The grouping operator ( ) controls the precedence of evaluation in expressions.

```
a + (b * c)
(a + b) * c;
```

NEW

You can use the new operator to create an instance of a user-defined object type or of one of the built-in object types.

```
const car1 = new Car("Eagle");
```

VOID

This operator allows evaluating expressions that produce a value into places where an expression that evaluates to undefined is desired.

```
void (2 == '2'); //undefined
```

COMMA OPERATOR

The comma operator (,) evaluates both of its operands and returns the value of the last operand.

```
let a, b, c;

a = b = 3, c = 4; // Returns 4 in console
console.log(a); // 3 (left-most)
```