Operators

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

There are different types of JavaScript operators:

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. Conditional Operators
7. Type Operators

Arithmetic Operators

Oper	Name	Example	Results
+	Addition	x = y + 2	y=5, x=7
-	Subtraction	x=y-2	y=5, x=3
*	Multiplication	x=y*2	y=5, x=10
**	Exponentiation ES2016	x=y**2	y=5, x=25
/	Division	x = y / 2	y=5, x=2.5
%	Remainder	x = y % 2	y=5, x=1
++	Pre increment	x = ++y	y=6, x=6
++	Post increment	x = y++	y=6, x=5
--	Pre decrement	x = --y	y=4, x=4
--	Post decrement	x = y--	y=4, x=5

```
» y = 6
← 6

» x=++y
← 7

» x
← 7

» y = 6
← 6

» x = y++
← 6
```

Notes on Arithmetic Operators

Modulo is extensively used when we have to check for odd or even numbers.

What is the difference between post-increment and pre-increment?

Ans:

Pre-increment: first it increments then it assigns.
Post-increment: first it assigns, then it increments.

X = y++
X= ++y
Value of y will be incremented in both the cases.
The change is in the assignment.

```
» y = 6
← 6
» x = y++
← 6
» console.log(x, y)
  6 7
← undefined
» y = 6
← 6
» x = ++y
← 7
» console.log(x, y)
  7 7
```

Assignment Operators

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

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

Assigment to a property of a JSON object

```
/* Assigning to a string variable */
let name = "Ashish";

/* Assigning to a numeric variable */
let age = 7

/* Assigning to a property of a JSON object at the time of definition */
let person = {
    "name": "Alpha",
    "nationality": "Indian"
};

/* Assigning to a property of a JSON object through dot operator */
person.name = "Beta";
```

Comparison Operators

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

Oper	Name	Comparing	Returns
==	equal to	x == 8	false
==	equal to	x == 5	true
===	equal value and type	x === "5"	false
===	equal value and type	x === 5	true
!=	not equal	x != 8	true
!==	not equal value or type	x !== "5"	true
!==	not equal value or type	x !== 5	false
>	greater than	x > 8	false
<	less than	x < 8	true
>=	greater or equal to	x >= 8	false
<=	less or equal to	x <= 8	*true*

Notes on Comparison Operators

```
» x = "5"
← "5"
» x == 5
← true
» x == 6
← false
» x
← "5"
» x === 5
← false
» x === "5"
← true
»
```

Logical Operators

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

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

Check if a person is eligible work or not?

Age is that factor. If age is < 18, then the person is too young.
If age > 60, then the person is too old.

```
» age = 55
← 55
» age >= 18 && age <= 60
← true
» age = 17.5
← 17.5
» age >= 18 && age <= 60
← false
```

Check eligibility for a scholarship

If the grade of a person is ‘A’ or ‘A+’, the person is eligible for a scholarship.

```
» grade = 'B'
← "B"
» grade === 'A' || grade === 'A+'
← false
» grade = 'A'
← "A"
» grade === 'A' || grade === 'A+'
← true
```

Check non-eligibility for scholarship

If I just say “grade not equal to A or A+”.
Or this is good “grade == B or grade == C or grade == D or grade == E or grade == F or grade == ‘B+’ ...” This will be very complex.

Eligibility condition is: grade === 'A' || grade === 'A+'
Non eligibility condition is:

```
» x = true
← true
» !x
← false
» y = false
← false
» !y
← true
» grade = 'B'
← "B"
» !(grade === 'A' || grade === 'A+')
← true
» grade === 'A' || grade === 'A+'
← false
```

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
&	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
>>>	Unsigned right	x = 5 >>> 1	0101 >>> 1	0010	2

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)

Truth Tables For AND (&), OR (|) and XOR (^)

x	y	&
0	0	0
0	1	0
1	0	0
1	1	1

x	y	|
0	0	0
0	1	1
1	0	1
1	1	1

x	y	^
0	0	0
0	1	1
1	0	1
1	1	0

Bitwise AND

```
> x = 5
< 5
> x.toString(2)
< '101'
```

x	1	0	1
y	1	1	1
x & y	1	0	1

```
> y = 7
< 7
> y.toString(2)
< '111'
> x & y
< 5
```

Bitwise OR

```
> x = 5
<· 5
> y = 7
<· 7
> x|y
<· 7
> x.toString(2)
<· '101'
> y.toString(2)
<· '111'
```

x	1	0	1
y	1	1	1
x | y	1	1	1

Another example of Bitwise AND and OR

```
> x = 10
< 10
> x.toString(2)
< '1010'
> y = 7
< 7
> y.toString(2)
< '111'
> x | y
< 15
> z = x|y
< 15
> z.toString(2)
< '1111'
```

x	1	0	1	0
y		1	1	1
|	1	1	1	1

```
> z=x&y
< 2
> z.toString(2)
< '10'
```

x	1	0	1	0
y		1	1	1
&	0	0	1	0