Operators

Operators in JavaScript

Arithmetic Operators

Used for mathematical calculations.

```
let a = 10;
let b = 20;

let sum = a + b;
let difference = a - b;
let product = a * b;
let quotient = a / b;
let remainder = a % b; // Modulus
let power = a ** b;

console.log(sum); // 30
console.log(difference); // -10
console.log(product); // 200
console.log(quotient); // 0.5
console.log(remainder); // 10
console.log(power); // 100000
```

Comparison Operators

Used to compare two values.

- `==` (Equal to) - Checks only data.
- `===` (Strict equal to) - Checks data and data type.

- `!=` (Not equal to).
- `>` (Greater than).
- `<` (Less than).
- `>=` (Greater than or equal to).
- `<=` (Less than or equal to).

```
let x = 10;
let y = 10;

console.log(x == y); // true
console.log(x === y); // true
console.log(x != y); // false
console.log(x > y); // false
console.log(x < y); // false
console.log(x >= y); // true
console.log(x <= y); // true
```

Logical Operators

Used to combine multiple conditions.

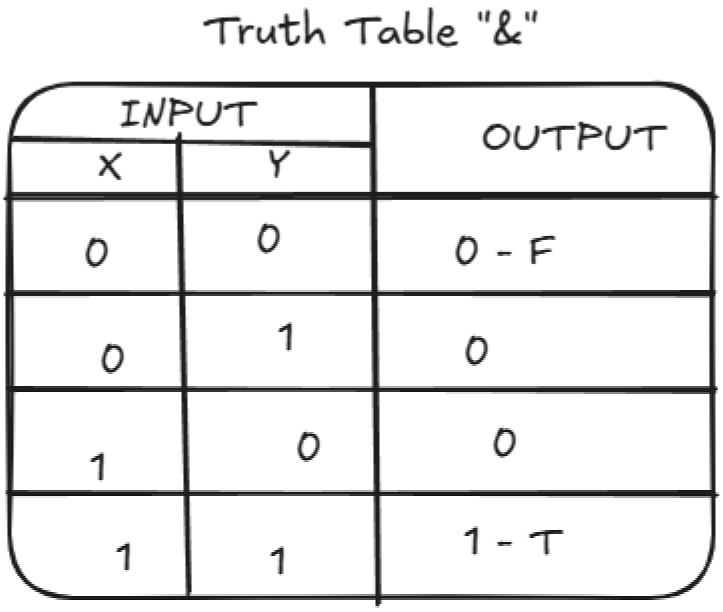
Operator	Description	Example	Result
`&&`	Logical AND	`true && false`	`false`
`||`	Logical OR	`true && false`	`true`
`!`	Logical NOT	`!true`	`false`

```
console.log(true && false); // false
console.log(true || false); // true
console.log(!false); // true
```

Bitwise Operators

1. **Bitwise AND (`&`)**

- Compares each bit of two numbers.
- The result is `1` if both bits are `1`; otherwise, it's `0`.



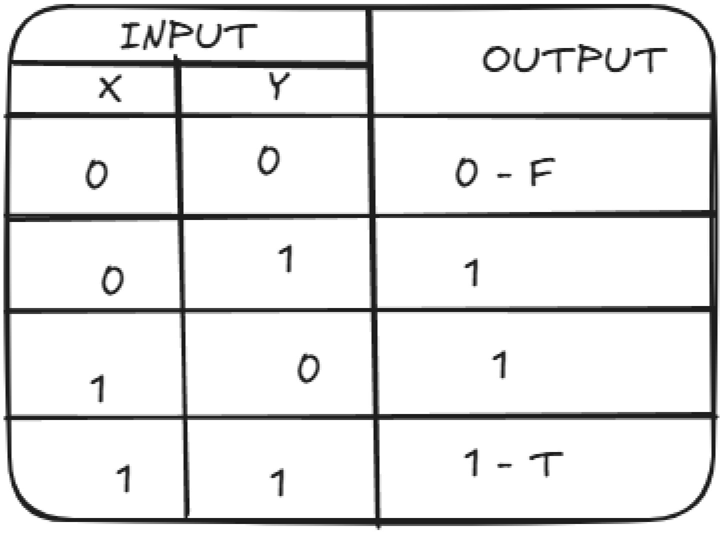
Truth Table "&"

INPUT		OUTPUT
X	Y	
0	0	0 - F
0	1	0
1	0	0
1	1	1 - T

```
X & 0 = 0
X & 1 = X
```

2. **Bitwise OR (`|`)**
    - Compares each bit of two numbers.
    - The result is `1` if at least one of the bits is `1`; otherwise, it's `0`.

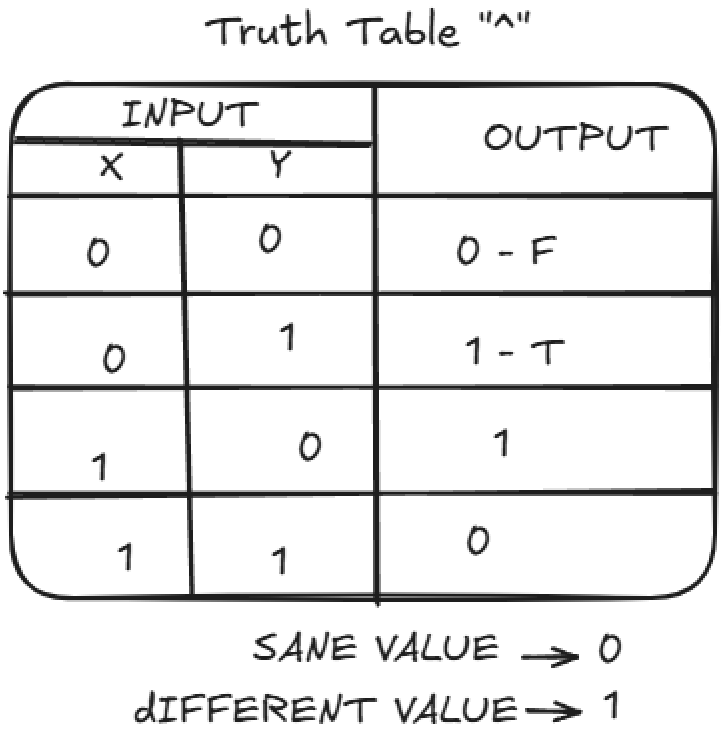
Truth Table "|"



INPUT		OUTPUT
X	Y	
0	0	0 - F
0	1	1
1	0	1
1	1	1 - T

```
X | 0 = X
X | 1 = 1
```

3. **Bitwise XOR (`^`)**
    - Compares each bit of two numbers.
    - The result is `1` if the bits are different; otherwise, it's `0`.



Truth Table "^"

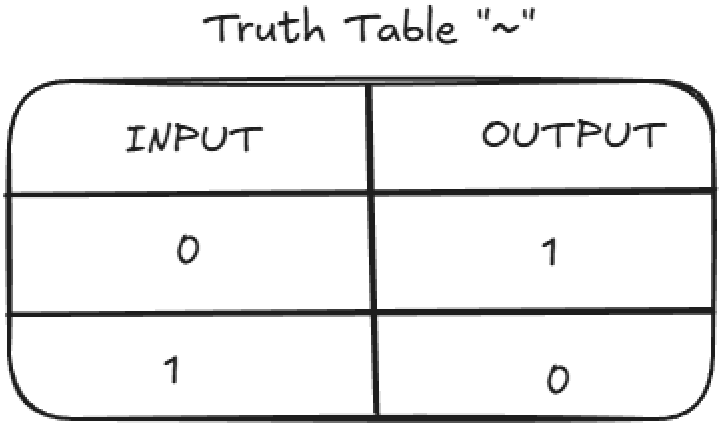
INPUT		OUTPUT
X	Y	
0	0	0 - F
0	1	1 - T
1	0	1
1	1	0

SANE VALUE → 0

dIFFERENT VALUE → 1

```
X ^ 0 = X
X ^ 1 = ~X
X ^ X = 0
```

4. **Bitwise NOT (`~`) - Negation [1's compliment]**
   - Inverts all bits (`1` becomes `0`, and `0` becomes `1`).



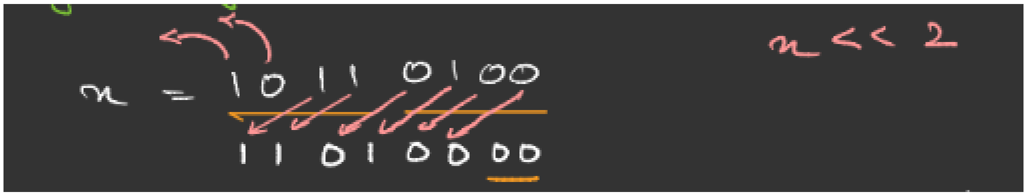
Truth Table "~"

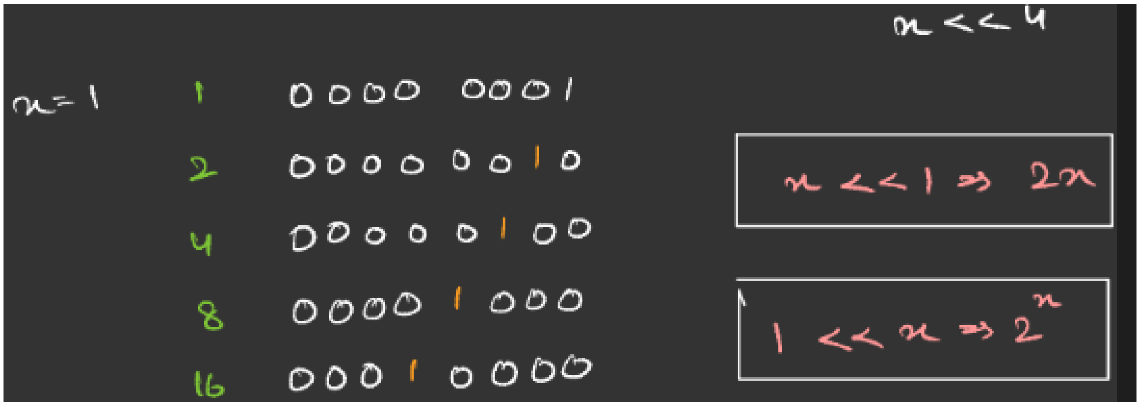
INPUT	OUTPUT
0	1
1	0

```
~0 = 1
~1 = 0
```

5. **Left Shift (`<<`)**
    - Shifts bits to the left, filling with zeros on the right.
    - `x<<n` multiplies x by 2^n.
    - Example:

        `0101<<1 = 1010`

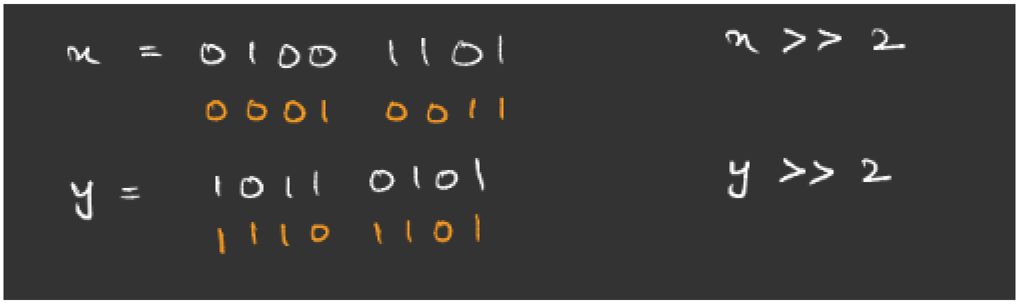




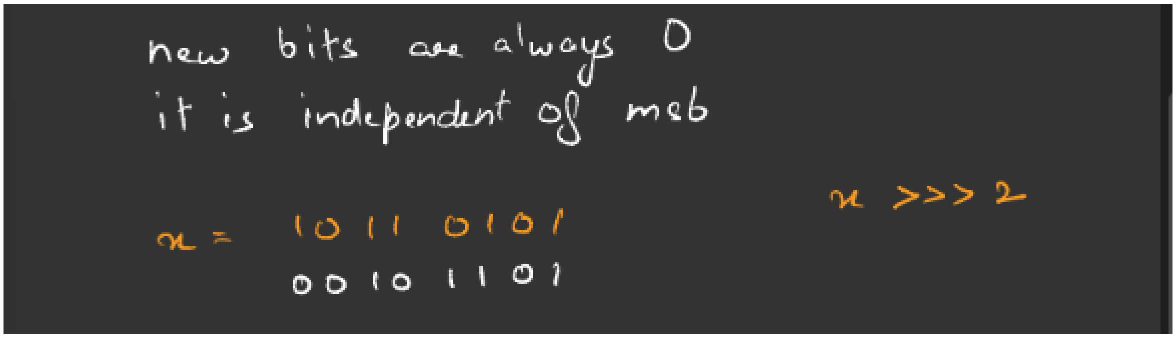
6. **Right Shift (`>>`)**

    New bits are always is independent of msb

    - Shifts bits to the right, filling with zeros (logical shift) or the sign bit (arithmetic shift).
    - `x>>n` divides x by 2^n.
    - Example: `0101>>1 = 0010`



Triple right shift (`>>>`)



new bits are always 0
it is independent of msb

$x >>> 2$

$x =$ 1011 0101
0010 1101