JavaScript Operators

There are different types of JavaScript operators:

1. Arithmetic Operators
2. Assignment Operators
3. Comparison Operators
4. Logical Operators
5. Bitwise Operators
6. String Operators
7. Ternary Operators
8. Type Operators

Arithmetic Operators:

Arithmetic operators are used to perform basic mathematical operations.

1. Addition (+)
   ```
   let sum = 5 + 3; // 8
   ```

2. Subtraction(-)
   ```
   let difference = 5 - 3; // 2
   ```

3. Multiplication(*)
   ```
   let product = 5 * 3; // 15
   ```

4. Division(/)
   ```
   let quotient = 5 / 3; // 1.6666666666666667
   ```

5. Modulus(%)
   ```
   let remainder = 5 % 3; // 2
   ```

6. Increment(++)
   ```
   let x = 5;
   x++; // x is now 6
   ```

7. Decrement(--)
   ```
   let y = 5;
   y--; // y is now 4
   ```

Assignment Operators:

Assignment operators are used to assign values to variables.

1. Assignment (=)
   ```
   let z = 10;
   ```

2. Addition Assignment(+=)
   ```
   z += 5; // z is now 15
   ```

3. Subtraction Assignment (-=)
   z -= 3; // z is now 12

4. Multiplication Assignment (*=)
   z *= 2; // z is now 24

5. Division Assignment (/=)
   z /= 4; // z is now 6

6. Modulus Assignment (%=)
   z %= 4; // z is now 2

7. Power Assignment(**=)
   z **= 2; // z is now 4

Comparison Operators:

Comparison operators are used to compare two values.

1. Equal (==)
   let x = 5;
   let y = 3;
   let isEqual = (x == 5); // true

2. Strict Equal (===)
   let isStrictEqual = (x === '5'); // false (different types)

3. Not Equal (!=)
   let isNotEqual = (x != 8); // true

4. Strict Not Equal (!==)
   let isStrictNotEqual = (x !== '5'); // true (different types)

5. Greater Than (>)
   let isGreaterThan = (x > 2); // true

6. Less Than (<)
   let isLessThan = (y < 5); // true

7. Greater Than or Equal (>=)
   let isGreaterThanOrEqual = (x >= 5); // true

8. Less Than or Equal (<=)
   let isLessThanOrEqual = (y <= 2); // true

Logical Operators

Logical operators are used to combine multiple conditions.

1. Logical AND (&&)

```
let x = 5;
let y = 3;
let andResult = (x > 2 && y < 5); // true
```

2. Logical OR (||)

```
let orResult = (x > 5 || y < 5); // true
```

3. Logical NOT (!)

```
let notResult = !(x > 2); // false
```

Bitwise Operators ---------- (B)

Bitwise operators perform operations on the binary representations of numbers.

1. AND (&)

```
let andBitwise = 5 & 1; // 1 (0101 & 0001)
```

2. OR (|)

```
let orBitwise = 5 | 1; // 5 (0101 | 0001)
```

3. NOT (~)

```
let notBitwise = ~5; // -6 (~0101)  ---------- (A)
```

4. XOR (^)

```
let xorBitwise = 5 ^ 1; // 4 (0101 ^ 0001)
```

5. Left Shift (<<)

```
let leftShift = 5 << 1; // 10 (0101 << 1)
```

6. Right Shift (>>)

```
let rightShift = 5 >> 1; // 2 (0101 >> 1)
```

7. Unsigned Right Shift (>>>)

```
let unsignedRightShift = 5 >>> 1; // 2 (0101 >>> 1)
```

String Operators

String operators are used to manipulate string values.

1. Concatenation (+):

```
let greeting = "Hello, " + "world!"; // "Hello, world!"
let x = 5 + 5; // 10
let y = "5" + 5; // 55   ---------- (C)
let z = "Hello" + 5; // Hello5
```

2. Concatenation Assignment (+=)
   let greet = "Hello";
   greet += ", world!"; // "Hello, world!"

Ternary Operator
The ternary operator (? :) is a shorthand for an if-else statement. It takes three operands: a condition, an expression to execute if the condition is true, and an expression to execute if the condition is false.
Syntax: condition ? ExpressionIf True : ExpressionIf False

```
let age = 18;
let canVote = (age >= 18) ? "Yes" : "No"; // "Yes"
```

Type Operators

Type operators are used to check the type of a variable or convert values to different types.
typeof: Returns a string indicating the type of the unevaluated operand.

```
let num = 42;
let typeOfNum = typeof num; // "number"
```

instanceof: Tests whether an object is an instance of a constructor or a class.

```
let date = new Date();
let isDate = date instanceof Date; // true
```

IMPORTANT:

A) The bitwise NOT operator (~) inverts each bit of its operand. To understand why ~5 results in -6, let's break it down step by step:
Convert the number to binary: The number 5 in binary (using 32-bit representation) is:
00000000000000000000000000000101

Invert the bits: Apply the bitwise NOT operator, which flips every bit (0 becomes 1 and 1 becomes 0):
11111111111111111111111111111010

Convert the result back to decimal: The result is a binary number that represents a negative value in two's complement form. To understand this, we need to convert it to its decimal equivalent.
Invert the bits again to find the positive counterpart:
00000000000000000000000000000101
Add 1 to the inverted bits:
00000000000000000000000000000110
This binary number (00000000000000000000000000000110) is 6 in decimal.

Add the negative sign: Since we are working with two's complement, the original number (11111111111111111111111111111010) represents -6.
Therefore, ~5 results in -6.

B) Bit operators work on 32 bits numbers.

C) If you add a number and a string, the result will be a string!