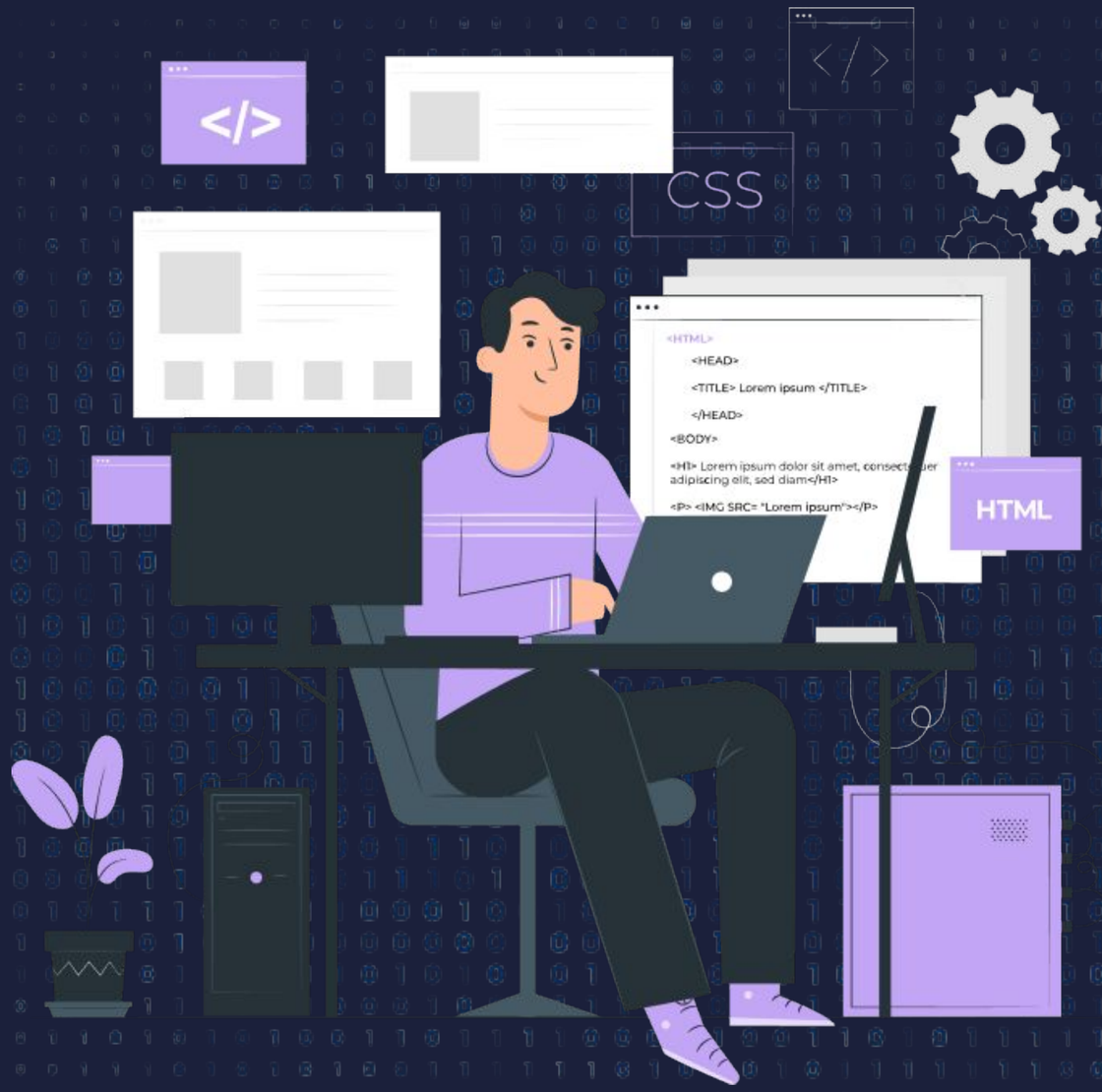
Operators and its Type



Topics Covered

- Operators & Operands.
- Types of Operators in JavaScript.

Operators & Operands.

Manipulating Data, by performing various mathematical operations is done through operators.

```
5 + 3
```

Here, we have to add **5 and 3 (operands)** to get the final value. So we are using **+ (operator)** to add these two values. And the final value is 8.

Assignment Operators

Used to assign values to variables.

```
var name = "PW Skills"; // name = "PW Skills"
```

We also have shorthand assignment operators to perform perform operation like add, subtract and then assigns value.

```
// Shorthand Assignment Operators

// 1. Add & Assign
var students = 100; // students = 100
students += 50; // students = 150

// 2. Subtract & Assign
var students = 100; // students = 100
students -= 50; // students = 50

// 3. Multiply & Assign
var students = 100; // students = 100
students *= 50; // students = 5000

// 4. Divide & Assign
var students = 100; // students = 100
students /= 50; // students = 2

// 5. Modulus & Assign
var students = 100; // students = 100
students %= 50; // students = 0

// 6. Exponential & Assign
var students = 100; // students = 100
students **= 2; // students = 10000
```

Arithmetic Operators

To do mathematical operations like addition, subtraction, multiplication, division, etc.

- Addition (+): Adds two values together.
- Subtraction (-): Subtracts one value from another.
- Multiplication (*): Multiplies two values together.
- Division (/): Divides one value by another.
- Modulus(%): Returns the remainder of a division operation.
- Exponentiation(**): raises to the power of.
- Increment Operator(++): Increases the value by 1.
- Decrement Operator(--): Decreases the value by 1.

Examples

```
// Arithmetic Operators

var num1 = 100;
var num2 = 2;

// 1. Addition
var result = num1 + num2; // 102

// 2. Subtraction
var result = num1 - num2; // 98

// 3. Multiplication
var result = num1 * num2; // 200

// 4. Division
var result = num1 / num2; // 50

// 5. Modulus
var result = num1 % num2; // 0
```

```
// 6. Exponential
var result = num1 ** num2; // 10000

// 7. Pre Increment
var num = 10;
var result = ++num; // num = 11 ; result = 11

// 8. Post Increment
var num = 10;
var result = num++; // result = 10 ; num = 11

// 9. Pre Decrement
var num = 10;
var result = --num; // num = 9 ; result = 9

// 10. Post Decrement
var num = 10;
var result = num--; // result = 10 ; num = 9
```

Relational Operators

- Equal (==): Compares two values for equality, returns true if they are equal and false if they are not.
- Strict equal (===): Compares two values for equality and type, returns true if they are equal and of the same type, and false if they are not.
- Not equal (!=): Compares two values for inequality, returns true if they are not equal and false if they are.
- Strict not equal (!==): Compares two values for inequality or type, returns true if they are not equal or not of the same type, and false if they are.
- Greater than (>): Compares two values, returns true if the left operand is greater than the right operand, and false otherwise.
- Greater than or equal to (>=): Compares two values, returns true if the left operand is greater than or equal to the right operand and false otherwise.
- Less than (<): Compares two values, returns true if the left operand is less than the right operand and false otherwise.
- Less than or equal to (<=): Compares two values, returns true if the left operand is less than or equal to the right operand and false otherwise.

Examples

```
// Comparison Operators
var num1 = 10;
var num2 = 20;
var num3 = 10;

var str1 = "10";
var str2 = "20";

// 1. Equal to
var result = num1 == num2; // false
var result = num1 == num3; // true

// 2. Strict Equal
var result = num1 === num3; // true

var result = num1 === str1; // false

// 3. Not equal
var result = num1 != num2; // true

var result = num1 != num3; // false
```

```
// 4. Strict Not Equal
var result = num1 !== num3; // false

var result = num1 !== str1; // true

// 5. Greater than
var result = num1 > num3; // false
var result = num2 > num3; // true

// 6. Greater than or Equal to
var result = num1 >= num3; // true

var result = num2 >= num3; // true

// 7. Lesser than
var result = num1 < num3; // false

var result = num2 < num3; // false

// 8. Lesser than or Equal to
var result = num1 <= num3; // true
var result = num2 <= num3; // false
```

Note: It is always advisable to use strict equal(===) and strict not equal(!==), for equality because it checks types also, and makes sure that our code is safe from unexpected behaviors due to type mismatches.

Logical Operators

Perform logical operations and return a boolean value, either true or false.

Logical AND (&&): This operator returns true if both operands are true, and false otherwise. It is a logical first operator, meaning that if the first operand is false, the second operand is not evaluated.

operand 1	operand 2	operand1 && operand 2
true	true	**true**
false	true	false
true	false	false
false	false	false

Logical OR (||): This operator returns true if at least one of the operands is true, and false otherwise. Like && operator, it is also a short-circuit operator.

operand 1	operand 2	operand1 && operand 2
true	true	true
false	true	true
true	false	true
false	false	**false**

Logical NOT (!): This operator inverts the Boolean value of the operand. If the operand is true, the operator returns false, and if the operand is false, the operator returns true.

operand	!operand
true	false
false	true

Examples

```
// Logical Operators

var num1 = 10;
var num2 = 20;
var num3 = 10;

// 1. Logical AND
var result = num1 >= num3 && num1 == num3; // true
var result = num1 >= num2 && num1 == num3; // false

// 2. Logical OR
var result = num1 >= num3 || num1 == num3; // true
var result = num1 >= num2 || num1 == num3; // true
var result = num1 >= num2 || num1 > num3; // false

// 3. Logical NOT
var result = num1 == num3; // true
var result = !(num1 == num3); // false
```

Bitwise Operators

Bitwise operators perform operations on binary representations of numbers.

- Bitwise AND (&): This operator compares each bit of the first operand to the corresponding bit of the second operand, and if both bits are 1, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
- Bitwise OR (|): This operator compares each bit of the first operand to the corresponding bit of the second operand, and if at least one of the bits is 1, the corresponding result bit is set to 1.
- Bitwise XOR (^): This operator compares each bit of the first operand to the corresponding bit of the second operand, and if the bits are different, the corresponding result bit is set to 1. Otherwise, the corresponding result bit is set to 0.
- Bitwise NOT (~): This operator inverts all the bits of the operand, effectively swapping 1s for 0s and 0s for 1s.
- Left Shift (<<): This operator shifts the bits of the operand to the left by the specified number of places, adding zeros to the right.
- Right Shift (>>): This operator shifts the bits of the operand to the right by the specified number of places, discarding the bits shifted out.

Examples

```
//1. AND (&) operator:
let a = 10;  // binary: 1010
let b = 6;   // binary: 0110
let result = a & b;  // binary: 0010 (decimal: 2)

// 2.OR (|) operator:
let a = 10;  // binary: 1010
let b = 6;   // binary: 0110
let result = a | b;  // binary: 1110 (decimal: 14)

// 3.XOR (^) operator:
let a = 10;  // binary: 1010
let b = 6;   // binary: 0110
let result = a ^ b;  // binary: 1100 (decimal: 12)
```

```
// 4.NOT (~) operator:
let a = 10;  // binary: 00000...1010(32
bits)
let result = ~a;  // binary:
11111...0101(32 bits) (decimal: -11)

// 5.Left shift (<<) operator:
let a = 5;   // binary: 0101
let result = a << 2;  // binary: 010100
(decimal: 20)

// 6.Right shift (>>) operator:
let a = 20;  // binary: 010100
let result = a >> 2;  // binary: 0101
(decimal: 5)
```

An Example of Bitwise Operator

1. Double any number : Left shift by 1
2. Half any number : Right shift by 1

```
// Double any number : Left shift by 1
10 << 1  // 20
22 << 1 // 44

//Half any number : Right shift by 1
66 >> 1  // 33
100 << 1 // 50
```


SKILLS
THANK YOU