Operators:

Operators are symbols used to perform operations on variables and values

- **Arithmetic Operators**: Used to perform arithmetic operations

```
let x = 10;
let y = 5;
let addition = x + y; // Addition
let subtraction = x - y; // Subtraction
let multiplication = x * y; // Multiplication
let division = x / y; // Division
let modulus = x % y; // Modulus (remainder)
let increment = x++; // Increment
let decrement = y--; // Decrement
```

- **Assignment Operators**: Used to assign values to variables.

```
let x = 10;
x += 5; // Equivalent to x = x + 5
x -= 5; // Equivalent to x = x - 5
x *= 5; // Equivalent to x = x * 5
x /= 5; // Equivalent to x = x / 5
```

- **Comparison Operators**: Used to compare values. They return a boolean value - true or false.

```
let a = 10;
let b = 5;
console.log(a > b); // Greater than
```

```
console.log(a < b); // Less than
console.log(a >= b); // Greater than or equal to
console.log(a <= b); // Less than or equal to
console.log(a === b); // Equal to (strict equality)
console.log(a !== b); // Not equal to (strict inequality)
```

- **Logical Operators**: Used to combine or manipulate boolean values.

```
let p = true;
let q = false;
console.log(p && q); // AND
console.log(p || q); // OR
console.log(!p); // NOT
```

//logical end

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

//logical or

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

- **Ternary Operator (Conditional Operator):** Used to assign a value to a variable based on a condition.

condition ? expressionIfTrue : expressionIfFalse;

```
let age = 20;
let status = age >= 18 ? "Adult" : "Minor";
console.log(status); // Output: 'Adult'
```

Nullish Coalescing Operator (??)

It is a logical operator that returns its right-hand operand when its left-hand operand is **null** or **undefined**, and otherwise returns its left-hand operand. It's useful for providing default values in expressions without overriding valid falsy values like 0, NaN, or ''.

Syntax

let result = expression1 ?? expression2;

```
//case - 1
let name1 = null;
let defaultName1 = name1 ?? "yes it is null or undefined";
console.log(defaultName1); // Output: 'yes it is null or undefined'

//case - 2
let name2 = undefined;
let defaultName2 = name2 ?? "yes it is null or undefined";
console.log(defaultName2); // Output: 'yes it is null or undefined'

//case - 3
let name3 = "hello world";
let defaultName3 = name3 ?? "yes it is null or undefined";
console.log(defaultName3); // Output: hello world
```

Practical Applications

Handling Optional Function Parameters:

```
    function hello(a){

      var b=a ?? "man";
      console.log("hello " + b);
    }
    hello();// we are not any value but still we are getting value because of Nullish Coalescing Operator
    hello("teja")
```

Optional Chaining Operator (?.)

It is a powerful tool that allows for safe navigation through nested object properties, functions, and arrays. It prevents runtime errors that occur when accessing properties of null or undefined objects, returning undefined instead of throwing an error.

Syntax

let result = object?.property;

let result = object?.[property];

let result = object?.method?.();

```

```

```
// wihout using optional chaining operator
  var obj = {
    name: "tej",
    state: {
      name: "ap",
    },
  };
  console.log(obj.obj.state); //it throws an error

  // with using optional chaining operator
  var obj = {
    name: "tej",
    state: {
      name: "ap",
    },
  };

console.log(obj.obj?.state); //undefined instead of error
```

Bitwise operator

Bitwise operators in JavaScript perform operations on binary representations of numbers. These operators treat their operands as a sequence of 32 bits (zeros and ones) and perform operations at the binary level.

AND (&)

- Sets each bit to 1 if both corresponding bits are 1.

let a = 5; // 0101

let b = 3; // 0011

let result = a & b; // 0001 (1 in decimal)

OR (|)

- Sets each bit to 1 if at least one of the corresponding bits is 1.

let a = 5; // 0101

let b = 3; // 0011

let result = a | b; // 0111 (7 in decimal)

XOR (^)

- Sets each bit to 1 if only one of the corresponding bits is 1.

let a = 5; // 0101

let b = 3; // 0011

let result = a ^ b; // 0110 (6 in decimal)

String operators

It is used to perform operations on string values. The primary string operator is the concatenation operator (+), but other operations involving strings include template literals, comparison operators, and methods available on string objects.

Concatenation Operator (+)

The concatenation operator combines two or more strings into a single string.

```
let greeting = "Hello, " + "world!";
console.log(greeting); // Outputs: "Hello, world!"
```