Q1. Explain the role of operators in JavaScript. Why are they essential in programming ?

Answer - In JavaScript, operators play a crucial role in performing operations on data. They are essential in programming because they allow you to manipulate and evaluate values, making it possible to create functional and dynamic applications. Here's a breakdown of the different types of operators and their roles:

1. Arithmetic Operators

Purpose: Perform mathematical operations.

Examples: + (addition), - (subtraction), * (multiplication), / (division), % (modulus).

Usage: These operators are used to calculate values, such as summing two numbers or finding the remainder of a division.

2. Assignment Operators

Purpose: Assign values to variables.

Examples: = (basic assignment), += (add and assign), -= (subtract and assign), *= (multiply and assign), /= (divide and assign).

Usage: These operators simplify the process of updating variables. For example, x += 5 is a shorthand for x = x + 5.

3. Comparison Operators

Purpose: Compare values and return a Boolean (true or false).

Examples: == (equality), === (strict equality), != (inequality), !== (strict inequality), > (greater than), < (less than), >= (greater than or equal to), <= (less than or equal to).

Usage: These operators are used in conditional statements to control the flow of the program based on comparisons.

4. Logical Operators

Purpose: Combine or invert Boolean values.

Examples: && (logical AND), || (logical OR), ! (logical NOT).

Usage: Useful in constructing complex conditions in control flow statements. For instance, if (x > 10 && y < 5) checks if both conditions are true.

5. Unary Operators

Purpose: Operate on a single operand.

Examples: + (unary plus), - (unary minus), ++ (increment), -- (decrement), ! (logical NOT), typeof (type of).

Usage: They perform operations such as toggling a Boolean value or changing the sign of a number.

6. Bitwise Operators

Purpose: Perform operations on binary representations of integers.

Examples: & (AND), | (OR), ^ (XOR), ~ (NOT), << (left shift), >> (right shift), >>> (unsigned right shift).

Usage: Useful for low-level programming tasks like bit manipulation.

7. Ternary Operator

Purpose: A shorthand for an if-else statement.

Example: condition ? expr1 : expr2.

Usage: Provides a concise way to return one of two values based on a condition.

8. Type Operators

Purpose: Provide information about data types.

Examples: typeof (returns the type of a variable), instanceof (checks if an object is an instance of a class).

Usage: Helps in dynamic typing by determining the type of a value or checking if an object belongs to a specific type.

Why Operators Are Essential:

Data Manipulation: They allow for arithmetic calculations, comparisons, and logical operations on data, which are fundamental for creating meaningful and dynamic applications.

Control Flow: Operators are critical for controlling the flow of execution in your code through conditional statements and loops.

Efficiency: They enable concise and readable code, making it easier to implement and maintain complex logic.

Flexibility: Different operators can handle various data types and scenarios, providing versatility in how you process and use data.

Q2. Describe the categorization of operators in JavaScript based on their functionality. Provide examples for each category.

Answer - In JavaScript, operators are categorized based on their functionality. Here's a breakdown of the main categories:

1. Arithmetic Operators

These operators perform basic mathematical operations.

Addition (+): Adds two values.

```
let sum = 5 + 3; // 8
```

Subtraction (-): Subtracts one value from another.

```
let difference = 10 - 4; // 6
```

Multiplication (*): Multiplies two values.

```
let product = 7 * 6; // 42
```

Division (/): Divides one value by another.

```
let quotient = 20 / 4; // 5
```

Modulus (%): Returns the remainder of a division operation.

```
let remainder = 10 % 3; // 1
```

Exponentiation (**): Raises a number to the power of another number.

```
let power = 2 ** 3; // 8
```

Unary Plus (+): Converts a variable to a number.

```
let num = +"123"; // 123
```

Unary Minus (-): Negates a number.

let negative = -5; // -5

2. Comparison Operators

These operators compare two values and return a boolean result.

Equal to (==): Checks if two values are equal, with type coercion.

let isEqual = (5 == '5'); // true

Strict equal to (===): Checks if two values are equal, without type coercion.

let isStrictEqual = (5 === '5'); // false

Not equal to (!=): Checks if two values are not equal, with type coercion.

let isNotEqual = (5 != '5'); // false

Strict not equal to (!==): Checks if two values are not equal, without type coercion.

let isStrictNotEqual = (5 !== '5'); // true

Greater than (>): Checks if a value is greater than another.

let isGreater = (10 > 5); // true

Less than (<): Checks if a value is less than another.

let isLess = (3 < 8); // true

Greater than or equal to (>=): Checks if a value is greater than or equal to another.

let isGreaterOrEqual = (5 >= 5); // true

Less than or equal to (<=): Checks if a value is less than or equal to another.

let isLessOrEqual = (4 <= 7); // true

3. Logical Operators

These operators perform logical operations and return boolean values.

Logical AND (&&): Returns true if both operands are true.

```
let andResult = (true && false); // false
```

Logical OR (||): Returns true if at least one operand is true.

```
let orResult = (true || false); // true
```

Logical NOT (!): Negates the boolean value of an operand.

```
let notResult = !true; // false
```

4. Assignment Operators

These operators are used to assign values to variables.

Assignment (=): Assigns a value to a variable.

```
let x = 10; // 10
```

Addition assignment (+=): Adds a value to a variable and assigns the result.

```
x += 5; // x is now 15
```

Subtraction assignment (-=): Subtracts a value from a variable and assigns the result.

```
x -= 3; // x is now 12
```

Multiplication assignment (*=): Multiplies a variable by a value and assigns the result.

```
x *= 2; // x is now 24
```

Division assignment (/=): Divides a variable by a value and assigns the result.

```
x /= 4; // x is now 6
```

Modulus assignment (%=): Applies the modulus operator to a variable and assigns the result.

```
x %= 5; // x is now 1
```

Exponentiation assignment (**=): Raises a variable to the power of a value and assigns the result.

```
x **= 3; // x is now 1
```

5. String Operators

These operators are used specifically with strings.

Concatenation (+): Combines two strings.

```
let greeting = "Hello, " + "world!"; // "Hello, world!"
```

Concatenation assignment (+=): Appends a string to another string variable.

```
let message = "Hello";
message += ", world!"; // "Hello, world!"
```

6. Unary Operators

These operators work with a single operand.

Increment (++): Increases the value of a variable by one.

```
let count = 1;
count++; // 2
```

Decrement (--): Decreases the value of a variable by one.

```
let count = 2;
count--; // 1
```

Typeof (typeof): Returns the type of a variable.

```
let type = typeof 123; // "number"
```

Delete (delete): Deletes a property from an object.

```
let obj = { name: "Alice" };
delete obj.name; // true, obj is now {}
```

7. Conditional (Ternary) Operator

This operator evaluates a condition and returns one of two values based on the result.

Ternary Operator (? :):

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

8. Comma Operator

This operator allows multiple expressions to be evaluated in a single statement.

Comma Operator (,):

```
let a = (1, 2, 3); // a is 3
```

9. Type Conversion Operators

These operators are used to convert values from one type to another.

String Conversion (String()):

```
let numStr = String(123); // "123"
```

Number Conversion (Number()):

```
let num = Number("123"); // 123
```

Boolean Conversion (Boolean()):

```
let bool = Boolean(1); // true
```

Q3. Differentiate between unary, binary, and ternary operators in JavaScript. Give examples of each.

Answer –

1. Unary Operators:

These operators operate on a single operand.

Examples include:

- Increment (++): Increases the value of a variable by 1.
- Decrement (--): Decreases the value of a variable by 1.
- Unary Plus (+): Converts a value to a number.
- Unary Negation (-): Negates a number.
- Logical NOT (!): Inverts a boolean value.
- typeof: Returns the type of a variable.

Example:

```
let x = 5;

let y = -x; // Unary Negation

x++; // Increment

console.log(typeof x); // typeof operator
```

2. Binary Operators:

These operators operate on two operands.

Examples include:

- Arithmetic operators (+, -, ``, /, %)*: Perform arithmetic calculations.
- Comparison operators (==, !=, >, <, >=, <=): Compare two values.
- Logical operators (&&, ||): Perform logical operations.
- Assignment operators (=, +=, -=, `=`, /= etc.)*: Assign a value to a variable.

Example:

```
let a = 10;

let b = 5;

let c = a + b; // Addition

let d = a > b; // Comparison
```

3. Ternary Operator:

- This is the only operator in JavaScript that takes three operands.
- It is also known as the conditional operator.
- The syntax is: `condition ? expression1 : expression2`.

- If the condition is true, expression1 is evaluated, otherwise expression2 is evaluated.

Example:

```
let age = 18;

let message = age >= 18 ? "Adult" : "Minor";
```

Q4. Discuss the precedence and associativity of operators in JavaScript. Why is understanding these concepts important?

Answer –

Operator Precedence:

- Determines which operator is executed first when an expression contains multiple operators.
- Operators with higher precedence are executed before operators with lower precedence.
- For example, the multiplication operator (*) has higher precedence than the addition operator (+).

Operator Associativity:

- Determines the order of evaluation for operators with the same precedence.
- Most operators in JavaScript are left-associative, meaning they are evaluated from left to right.
- For example, in the expression a + b + c, the addition is performed from left to right: (a + b) + c.
- A few operators, like the assignment operator (=), are right-associative.

Why is understanding these concepts important?

- Correct Evaluation: It helps ensure that expressions are evaluated correctly. Without understanding precedence and associativity, you might expect an expression to evaluate in one way, when in reality it evaluates in another.
- Avoiding Bugs: Misinterpreting precedence can lead to subtle bugs in your code.

- Writing Readable Code: Code that follows the standard precedence rules is easier to read and understand.

Example:

```
let result = 2 + 3 * 4;
```

In this example, the multiplication operator (*) has higher precedence than the addition operator (+). Therefore, the expression is evaluated as 2 + (3 * 4), resulting in 14.

Q5. Write a JavaScript program that calculates the simple interest using the formula Simple interest = (principal * rate * time) / 100.

Answer –

```
function calculateSimpleInterest(principal, rate, time) {
        if (isNaN(principal) || isNaN(rate) || isNaN(time) || principal <= 0 || rate <= 0 || time <= 0) {
        return "Invalid input. Principal, rate, and time must be positive numbers.";
  }

let simpleInterest = (principal * rate * time) / 100;
  return simpleInterest;
}
let principal = 1000; // Principal amount in dollars
let rate = 5;         // Annual interest rate in percent
let time = 2;         // Time in years

let interest = calculateSimpleInterest(principal, rate, time);

console.log(`The simple interest is $${interest}`);
```

Q6. Write a Javascript program to calculate the Body Mass Index (BMI) using the formula BMI = weight (kg)/height * height.

Answer –

```
function calculateBMI(weight, height) {
        if (isNaN(weight) || isNaN(height) || weight <= 0 || height <= 0) {
                return "Invalid input. Weight and height must be positive
numbers.";
  }

 let heightSquared = height * height;

 let bmi = weight / heightSquared;

  return bmi;
}
let weight = 70;  // Weight in kilograms
let height = 1.75; // Height in meters
let bmi = calculateBMI(weight, height);
console.log(`Your BMI is ${bmi.toFixed(2)}`);

function getBMICategory(bmi) {
  if (bmi < 18.5) {
    return "Underweight";
  } else if (bmi >= 18.5 && bmi < 24.9) {
    return "Normal weight";
  } else if (bmi >= 25 && bmi < 29.9) {
    return "Overweight";
  } else {
    return "Obesity";
  }
}

let category = getBMICategory(bmi);
console.log(`BMI Category: ${category}`);
```

Q7. Write a program in JavaScript to calculate the area of a circle given its radius value of 10. Use appropriate arithmetic operators.

Answer –

```
        function calculateCircleArea(radius) {
                if (isNaN(radius) || radius <= 0) {
                return "Invalid input. Radius must be a positive number.";
  }

const pi = Math.PI;
let area = pi * radius * radius;
return area;
}

let radius = 10;
let area = calculateCircleArea(radius);
console.log(`The area of the circle with radius ${radius} is ${area.toFixed(2)}`);
```