Solution 1:

Operators in JavaScript are symbols that perform operations on operands. Operands can be variables, values, or expressions. Operators are essential in programming because they allow us to manipulate data and perform calculations.
They are commonly used to:

- Perform arithmetic operations.
- Assign values using assignment operators.
- Compare values using comparison operators.
- Perform logical operations.

Solution 2:

Operators in JavaScript are categorized based on their functionality into the following groups:

1. Arithmetic Operators

These operators perform mathematical operations on operands, which can be numbers or variables.

javascript
Copy code
```
// Addition (+)
const sum = 1 + 2; // sum is now equal to 3

// Subtraction (-)
const difference = 10 - 5; // difference is now equal to 5

// Multiplication (*)
const product = 3 * 4; // product is now equal to 12

// Division (/)
const quotient = 12 / 3; // quotient is now equal to 4

// Exponentiation (**)
const power = 2 ** 3; // power is now equal to 8

// Modulo (%)
const remainder = 10 % 3; // remainder is now equal to 1
```

2. Assignment Operators

These operators assign values to variables.

javascript
Copy code
```
// Simple assignment operator (=)
let myVariable = 10; // myVariable is equal to 10

// Addition assignment operator (+=)
myVariable += 5; // myVariable is now equal to 15
```

```
// Subtraction assignment operator (-=)
myVariable -= 5; // myVariable is now equal to 10

// Multiplication assignment operator (*=)
myVariable *= 2; // myVariable is now equal to 20

// Division assignment operator (/=)
myVariable /= 2; // myVariable is now equal to 10

// Modulo assignment operator (%=)
myVariable %= 3; // myVariable is now equal to 1
```

3. Comparison Operators: These operators compare two values and return a Boolean value (true or false).

```
// Equal to (==)
const isEqual = 10 == 10; // isEqual is now equal to true

// Not equal to (!=)
const isNotEqual = 10 != 10; // isNotEqual is now equal to false

// Greater than (>)
const isGreaterThan = 10 > 5; // isGreaterThan is now equal to true

// Less than (<)
const isLessThan = 10 < 5; // isLessThan is now equal to false

// Greater than or equal to (>=)
const isGreaterThanOrEqual = 10 >= 10; // isGreaterThanOrEqual is now equal to

true
// Less than or equal to (<=)
const isLessThanOrEqual = 10 <= 5; // isLessThanOrEqual is now equal to false
```

4. Logical Operators: These operators perform logical operations on Boolean values.

```
// AND (&&)
const isAndTrue = true && true; // isAndTrue is now equal to true
const isAndFalse = true && false; // isAndFalse is now equal to false

// OR (||)
const isOrTrue = true || false; // isOrTrue is now equal to true
const isOrFalse = false || false; // isOrFalse is now equal to false

// NOT (!)
const isNotTrue = !true; // isNotTrue is now equal to false
const isNotFalse = !false; // isNotFalse is now equal to true
```

Solution 3:

Operators in JavaScript are categorized based on the number of operands they require, into the following groups:

1. Unary Operators

Unary operators operate on a single operand. These include:

- **Unary plus (+)**: Converts its operand into a number.
- **Unary negation (-)**: Negates its operand (i.e., converts it to a negative value).
- **Increment (++)**: Increases its operand by 1.
- **Decrement (--)**: Decreases its operand by 1.
- **Logical NOT (!)**: Converts its operand to a boolean and negates it.
- **typeof**: Returns the type of the operand.

2. Binary operators in JavaScript are:

+
-
*
/
%
==
===
!=
!==
>
<
>=
<=
||
&&

Solution 4:

Precedence refers to the order in which operators are evaluated when multiple operators are present in an expression. Operators with higher precedence are executed first. For example, in the expression 3 + 5 * 2, multiplication (*) has higher precedence than addition (+), so the multiplication is performed first, resulting in 3 + (5 * 2) = 13.

Associativity refers to the order in which operators of the same precedence are evaluated. Some operators associate from **left to right**, meaning they are evaluated in that direction. Others associate from **right to left**.

- **Left to right associativity**: Most operators, like arithmetic (+, -, *, /) and comparison (==, <, >) operators, are evaluated from left to right.
- **Right to left associativity**: Some operators, like the assignment operator (=), exponentiation (**), and the conditional (ternary) operator (? :), are evaluated from right to left.

Understanding operator precedence and associativity is important for several reasons:

1. **Correct Evaluation**: Understanding the order of precedence ensures that expressions are evaluated correctly, preventing potential errors or unexpected results.

Example:

javascript
Copy code
const result = 3 + 5 * 2; // Result: 13, because * has higher precedence than +

2. **Complex Expressions**: When working with complex expressions, knowing the precedence and associativity allows developers to write code that behaves as intended, making it easier to reason about and maintain.

   Example:

   javascript
   Copy code
   const result = (3 + 5) * 2; // Result: 16, parentheses override precedence

3. **Debugging**: In case of unexpected results, understanding operator precedence and associativity helps in debugging and identifying the root cause of issues.
4. **Code Optimization**: Knowledge of precedence and associativity can be used to optimize code by avoiding unnecessary parentheses and ensuring efficient computation.

Solution 5:

```
const principal = 1000;

const rate = 5;

const time = 2; // in years

const result = (principal * rate * time)/100;

console.log("Simple Interest =", result);
```

Solution 6:

```
const height = 160 // in cm

const weight = 55 // in kg

const BMI = weight/ height * height

console.log("BMI =", BMI)
```

Solution 7:

```
const radius = 10; // Radius of the circle

const pi = Math.PI; // Approximate value of pi

const area = pi * Math.pow(radius, 2);

console.log("Area of the circle:", area);
```