Math Operations

Agenda

- What mathematical operations are?
- Operators and operands
- Arithmetic Operators
- Arithmetic Assignments Operator
- Unary Operator
- Relation Operators
- Logical Operators
- Expressions in Java

What mathematical operations are?

What are mathematical operators?

Mathematical operators are symbols used to perform operations on operands. In programming languages like Java, they play a crucial role in performing arithmetic calculations, data manipulation, and directing computational logic.

Operators and operands

Operators and Operands

Each operator takes one, two or three operands:

1) a unary operator takes one operand

```
j++;
```

2) a binary operator takes two operands

```
i = j++;
```

3) a ternary operator requires three operands

```
i = (i>12) ? 1 : i++;
```

Assignment Operators

A binary operator:

variable = expression;

It assigns the value of the expression to the variable.

The types of the variable and expression must be compatible.

The value of the whole assignment expression is the value of the expression on the right, so it is possible to chain assignment expressions as follows:

```
int x, y, z;

x = y = z = 2;
```

Arithmetic Operators

Arithmetic Operators

Java supports various arithmetic operators for:

1) integer numbers

2) floating-point numbers

There are two kinds of arithmetic operators:

1) basic: addition, subtraction, multiplication, division and modulo

2) shortcut: arithmetic assignment

Table: Basic Arithmetic Operators

+	op1 + op2	Adds op1 and op2
-	op1 - op2	Subtract op1 and op2
*	op1 * op2	Multiplies op1 and op2
/	op1 / op2	Divides op1 and op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

Addition

Addition(+)

This operator is a binary operator and is used to add two operands.

For example:

```
class Addition {
        public static void main(String[] args)
        {
                // initializing variables
                int num1 = 10, num2 = 20, sum = 0;
                // Displaying num1 and num2
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                // adding num1 and num2
                sum = num1 + num2;
                System.out.println("The sum = " + sum);
        }}
```

Output:
num1=10
num2 = 20
The sum = 30

Subtraction

Subtraction(-):

This operator is a binary operator and is used to subtract two operands.

For example:

```
class Subtraction {
	public static void main(String[] args)
	{
		// initializing variables
		int num1 = 20, num2 = 10, sub = 0;

		// Displaying num1 and num2
		System.out.println("num1 = " + num1);
		System.out.println("num2 = " + num2);

		// subtracting num1 and num2
		sub = num1 - num2;
		System.out.println("Subtraction = " + sub);
	}
}
```

Output:
num1=20
num2 = 10
Subtraction = 10

Multiplication

Multiplication(*):

This operator is a binary operator and is used to multiply two operands.

For example:

```
class Multiplication {
	public static void main(String[] args)
	{
		// initializing variables
		int num1 = 20, num2 = 10, mult = 0;

		// Displaying num1 and num2
		System.out.println("num1 = " + num1);
		System.out.println("num2 = " + num2);

		// Multiplying num1 and num2
		mult = num1 * num2;
		System.out.println("Multiplication = " + mult);
	}
}
```

Output:
num1=20
num2 = 10
Multiplication = 200

Division

Division(/):

This is a binary operator that is used to divide the first operand(dividend) by the second operand(divisor) and give the quotient as a result

For example:

```
class Division {
	public static void main(String[] args)
	{
		// initializing variables
		int num1 = 20, num2 = 10, div = 0;

		// Displaying num1 and num2
		System.out.println("num1 = " + num1);
		System.out.println("num2 = " + num2);

		// Dividing num1 and num2
		div = num1 / num2;
		System.out.println("Division = " + div);
	}
}
```

Output:
num1=20
num2 = 10
Division = 2

Modulus

Modulus(%):

This is a binary operator that is used to return the remainder when the first operand(dividend) is divided by the second operand(divisor).

For example:

```
class Modulus {
	public static void main(String[] args)
	{
		// initializing variables
		int num1 = 5, num2 = 2, mod = 0;

		// Displaying num1 and num2
		System.out.println("num1 = " + num1);
		System.out.println("num2 = " + num2);

		// Remaindering num1 and num2
		mod = num1 % num2;
		System.out.println("Remainder = " + mod);
	}
}
```

Output:
num1=5
num2 = 2
Remainder = 1

Arithmetic Assignments Operators

Arithmetic Assignment Operators

These operators are used to assign values to a variable. The left side operand of the assignment operator is a variable, and the right side operand of the assignment operator is a value. The value on the right side must be of the same data type of the operand on the left side. Otherwise, the compiler will raise an error. This means that the assignment operators have right to left associativity, i.e., the value given on the right-hand side of the operator is assigned to the variable on the left. Therefore, the right-hand side value must be declared before using it or should be a constant. The general format of the assignment operator is:

```
variable operator value;
```

Types of Assignment Operators

The Assignment Operator is generally of two types. They are:

1. Simple Assignment Operator: The Simple Assignment Operator is used with the “=” sign where the left side consists of the operand and the right side consists of a value. The value of the right side must be of the same data type that has been defined on the left side.

2. Compound Assignment Operator: The Compound Operator is used where +,-,*, and / is used along with the = operator.

= Operator

(=) operator:

This is the most straightforward assignment operator, which is used to assign the value on the right to the variable on the left. This is the basic definition of an assignment operator and how it functions.

For example:

```
class Assignment {
        public static void main(String[] args)
        {
                // Declaring variables
                int num;
                String name;
                // Assigning values
                num = 10;
                name = "StartSteps";
                // Displaying the assigned values
                System.out.println("num is assigned: " + num);
                System.out.println("name is assigned: " + name);
        }
}
```

Output:
num is assigned: 10
name is assigned:StartSteps

+= Operator

(+=) operator:

This operator is a compound of '+' and '=' operators. It operates by adding the current value of the variable on the left to the value on the right and then assigning the result to the operand on the left.

For example:

```
class Assignment {
        public static void main(String[] args)
        {
                // Declaring variables
                int num1 = 10, num2 = 20;
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                // Adding & Assigning values
                num1 += num2;
                // Displaying the assigned values
                System.out.println("num1 = " + num1);
        }
}
```

Output:

num1 = 10

num2 = 20

num1 = 30

-= Operator

(-=) operator:

This operator is a compound of '-' and '=' operators. It operates by subtracting the variable's value on the right from the current value of the variable on the left and then assigning the result to the operand on the left.

For example:

```
class Assignment {
	public static void main(String[] args)
	{
		// Declaring variables
		int num1 = 10, num2 = 20;
		System.out.println("num1 = " + num1);
		System.out.println("num2 = " + num2);
		// Subtracting & Assigning values
		num1 -= num2;
		// Displaying the assigned values
		System.out.println("num1 = " + num1);
	}
}
```

Output:

num1 = 10

num2 = 20

num1 = -10

*= Operator

(*=) operator:

This operator is a compound of '*' and '=' operators. It operates by multiplying the current value of the variable on the left to the value on the right and then assigning the result to the operand on the left.

For example:

```
class Assignment {
        public static void main(String[] args)
        {
                // Declaring variables
                int num1 = 10, num2 = 20;
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                // Multiplying & Assigning values
                num1 *= num2;
                // Displaying the assigned values
                System.out.println("num1 = " + num1);
        }
}
```

Output:

num1 = 10

num2 = 20

num1 = 200

/= Operator

(/=) operator:

This operator is a compound of '/' and '=' operators. It operates by dividing the current value of the variable on the left by the value on the right and then assigning the quotient to the operand on the left.

For example:

```
class Assignment {
        public static void main(String[] args)
        {
                // Declaring variables
                int num1 = 20, num2 = 10;
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                // Dividing & Assigning values
                num1 /= num2;
                // Displaying the assigned values
                System.out.println("num1 = " + num1);
        }
}
```

Output:

num1 = 20

num2 = 10

num1 = 2

%= Operator

(%=) operator:

This operator is a compound of '%' and '=' operators. It operates by dividing the current value of the variable on the left by the value on the right and then assigning the remainder to the operand on the left.

For example:

```
class Assignment {
        public static void main(String[] args)
        {
                // Declaring variables
                int num1 = 5, num2 = 3;
                System.out.println("num1 = " + num1);
                System.out.println("num2 = " + num2);
                // Moduling & Assigning values
                num1 %= num2;
                // Displaying the assigned values
                System.out.println("num1 = " + num1);
        }
}
```

Output:

num1 = 5

num2 = 3

num1 = 2

Table: Arithmetic Assignments

+=	V += expr;	V = v + expr;
-=	v-=expr;	V = v - expr;
*=	v*=expr;	V = v * expr;
/=	v/=expr;	V = v / expr;
%=	v%=expr;	V = v % expr;

Unary Operator

Unary Operators

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Increment/Decrement Operators

Two unary operators:

1) ++ increments its operand by 1

2) -- decrements its operand by 1

The operand must be a numerical variable.

Each operation can appear in two versions:

- prefix version evaluates the value of the operand after performing the increment/decrement operation

- postfix version evaluates the value of the operand before performing the increment/decrement operation

Table: Increment/Decrement

++	v++;	Return value of v, then increment v
++	++v;	Increment v, then return its value
- -	v- -;	Return value of v, then decrement v
- -	- -v;	Decrement v , then return its value

Unary minus

Unary minus(-) This operator can be used to convert a positive value to a negative one.

For example:

```
// Main class
class StartSteps{
        // Main driver method
        public static void main(String[] args)
        {
                // Declaring a custom variable
                int n1 = 20;

                // Printing the above variable
                System.out.println("Number = " + n1);

                // Performing unary operation
                n1 = -n1;

                // Printing the above result number
                // after unary operation
                System.out.println("Result = " + n1);}}
```

Output:
Number = 20

Result = -20

‘NOT‘ Operator (!)

Operator(!) This is used to convert true to false or vice versa. Basically, it reverses the logical state of an operand.

For example:

```
// Main class
class StartSteps{
	// Main driver method
	public static void main(String[] args)
	{
		// Initializing variables
		boolean cond = true;
		int a = 10, b = 1;
		// Displaying values stored in above variables
		System.out.println("Cond is: " + cond);
		System.out.println("Var1 = " + a);
		System.out.println("Var2 = " + b);
		// Displaying values stored in above variables
		// after applying unary NOT operator
		System.out.println("Now cond is: " + !cond);
		System.out.println("!(a < b) = " + !(a < b));
		System.out.println("!(a > b) = " + !(a > b));
	}}
```

Output:

```
Cond is: true
Var1 = 10
Var2 = 1
Now cond is: false
!(a < b) = true
!(a > b) = false
```

Operator Increment

Increment(++) It is used to increment the value of an integer. It can be used in two separate ways:

1: Post-increment operator

When placed after the variable name, the value of the operand is incremented but the previous value is retained temporarily until the execution of this statement and it gets updated before the execution of the next statement.

Syntax : num++

Illustration: num = 5 —-----> num++ = 6

2: Pre-increment operator

When placed before the variable name, the operand's value is incremented instantly.

Syntax : ++num

Illustration : num = 5 —---------> ++num= 6

Operator Decrement

Decrement(–) It is used to decrement the value of an integer. It can be used in two separate ways:

1:Post-decrement operator

When placed after the variable name, the value of the operand is decremented but the previous values is retained temporarily until the execution of this statement and it gets updated before the execution of the next statement.

Syntax : num- -

Illustration: num = 5 —-----> num- - = 5 // Value will be decremented before execution of next statement.

2: Pre-decrement operator

When placed before the variable name, the operand's value is decremented instantly.

Syntax : - -num

Illustration : num = 5 —---------> - - num= 5 //output is 5, value is decremented before execution of next statement

Example program in Java that implements all basic unary operators for user input:


```
import java.util.Scanner;
public class UnaryOperators {
        public static void main(String[] args)
        {
                Scanner sc = new Scanner(System.in);
                // fro user inputs here is the code.
                // System.out.print("Enter a number: ");
                // int num = sc.nextInt();
                int num = 10;
                int result = +num;
                System.out.println("The value of result after unary plus is: "+ result);
                 result = -num;
                System.out.println("The value of result after unary minus is: "+ result);
                result = ++num;
                System.out.println("The value of result after pre-increment is: "+ result);
                result = num++;
                System.out.println("The value of result after post-increment is: "+ result);
                result = --num;
                System.out.println("The value of result after pre-decrement is: "+ result);
                result = num--;
                System.out.println("The value of result after post-decrement is: "+ result);}}
```

Output:

```
The value of result after unary plus is: 10
The value of result after unary minus is: -10
The value of result after pre-increment is: 11
The value of result after post-increment is: 11
The value of result after pre-decrement is: 11
The value of result after post-decrement is: 11
```

Break 10 minutes

Relation Operators

Relation Operators

Relational operators determine the relationship that one operand has to the other operand, specifically equality and ordering.

The outcome is always a value of type boolean.

They are most often used in branching and loop control statements.

Table: Relational Operators

==	Equals to	Apply to any type
!=	Not equal to	Apply to any type
>	Greater than	Apply to numerical types only
<	Less than	Apply to numerical types only
>=	Greater than or equal	Apply to numerical types only
<=	Less than or equal	Apply to numerical types only

‘Equal to‘ operator

‘Equal to’ operator (==) This operator is used to check whether the two given operands are equal or not. The operator returns true if the operand at the left-hand side is equal to the right-hand side, else false.

For example:

```
class StartSteps{

        // Main driver method
        public static void main(String[] args)
        {
                // Initializing variables
                int var1 = 5, var2 = 10, var3 = 5;
                // Displaying var1, var2, var3
                System.out.println("Var1 = " + var1);
                System.out.println("Var2 = " + var2);
                System.out.println("Var3 = " + var3);
                // Comparing var1 and var2 and
                // printing corresponding boolean value
                System.out.println("var1 == var2: "
                                                + (var1 == var2));
                // Comparing var1 and var3 and
                // printing corresponding boolean value
                System.out.println("var1 == var3: "
                                                + (var1 == var3));}}
```

Output:

```
Var1 = 5
Var2 = 10
Var3 = 5
var1 == var2: false
var1 == var3: true
```

‘Not Equal to‘ operator

‘Not equal to’ Operator(!=) This operator is used to check whether the two given operands are equal or not. It functions opposite to that of the equal-to-operator. It returns true if the operand at the left-hand side is not equal to the right-hand side, else false.
For example:

```
class StartSteps{
  // Main driver method
  public static void main(String[] args)
  {
    // Initializing variables
    int var1 = 5, var2 = 10, var3 = 5;
     // Displaying var1, var2, var3
    System.out.println("Var1 = " + var1);
    System.out.println("Var2 = " + var2);
    System.out.println("Var3 = " + var3);
    // Comparing var1 and var2 and
    // printing corresponding boolean value
    System.out.println("var1 != var2: "+ (var1 != var2));
    // Comparing var1 and var3 and
    // printing corresponding boolean value
    System.out.println("var1 != var3: "+ (var1 != var3));
}}
```

Output:
Var1 = 5
Var2 = 10
Var3 = 5
var1 != var2: true
var1 != var3: false

‘Greater than‘ operator

‘Greater than’ operator(>) This checks whether the first operand is greater than the second operand or not. The operator returns true when the operand at the left-hand side is greater than the right-hand side.
For example:

```
// Main class
class Startsteps{
  // Main driver method
  public static void main(String[] args)
  {
    // Initializing variables
    int var1 = 30, var2 = 20, var3 = 5;
    // Displaying var1, var2, var3
    System.out.println("Var1 = " + var1);
    System.out.println("Var2 = " + var2);
    System.out.println("Var3 = " + var3);
    // Comparing var1 and var2 and
    // printing corresponding boolean value
    System.out.println("var1 > var2: " + (var1 > var2));
     // Comparing var1 and var3 and
    // printing corresponding boolean value
    System.out.println("var3 > var1: " + (var3 >= var1));
 }}
```

Output:
Var1 = 20
Var2 = 20
Var3 = 5
var1 > var2: true
var3 > var1: false

‘Less than‘ operator

‘Less than’ Operator(<) This checks whether the first operand is less than the second operand or not. The operator returns true when the operand at the left-hand side is less than the right-hand side. It functions opposite to that of the greater-than operator.

For example:

```
class StartSteps{
  // Main driver method
  public static void main(String[] args)
  {
    // Initializing variables
    int var1 = 10, var2 = 20, var3 = 5;
    // Displaying var1, var2, var3
    System.out.println("Var1 = " + var1);
    System.out.println("Var2 = " + var2);
    System.out.println("Var3 = " + var3);
    // Comparing var1 and var2 and
    // printing corresponding boolean value
    System.out.println("var1 < var2: " + (var1 < var2));
    // Comparing var2 and var3 and
    // printing corresponding boolean value
    System.out.println("var2 < var3: " + (var2 < var3));
  }}
```

Output:

Var1 = 10

Var2 = 20

Var3 = 5

var1 < var2: true

var2 < var3: false

‘Greater than or equal to‘ operator

Greater than or equal to (>=) This checks whether the first operand is greater than or equal to the second operand or not. The operator returns true when the operand at the left-hand side is greater than or equal to the right-hand side.

For example:

```
// Main class
class StartSteps {
  // Main driver method
  public static void main(String[] args)
  {
    // Initializing variables
    int var1 = 20, var2 = 20, var3 = 10;
    // Displaying var1, var2, var3
    System.out.println("Var1 = " + var1);
    System.out.println("Var2 = " + var2);
    System.out.println("Var3 = " + var3);
    // Comparing var1 and var2 and
    // printing corresponding boolean value
    System.out.println("var1 >= var2: "+ (var1 >= var2));
     // Comparing var2 and var3 and
    // printing corresponding boolean value
    System.out.println("var2 >= var3: "+ (var2 >= var3));
}}
```

Output:

Var1 = 20

Var2 = 20

Var3 = 10

var1 >= var2: true

var2 >= var3: true

‘Less than or equal to‘ operator

Less than or equal to (<=) This checks whether the first operand is less than or equal to the second operand or not. The operator returns true when the operand at the left-hand side is less than or equal to the right-hand side.

For example:

```
class StartSteps{
  // Main driver method
  public static void main(String[] args)
  {
    // Initializing variables
    int var1 = 10, var2 = 10, var3 = 9;
    // Displaying var1, var2, var3
    System.out.println("Var1 = " + var1);
    System.out.println("Var2 = " + var2);
    System.out.println("Var3 = " + var3);
    // Comparing var1 and var2 and
    // printing corresponding boolean value
    System.out.println("var1 <= var2: "+ (var1 <= var2));
    // Comparing var2 and var3 and
    // printing corresponding boolean value
    System.out.println("var2 <= var3: "+ (var2 <= var3));
  }}
```

Output:

Var1 = 10

Var2 = 10

Var3 = 9

var1 <= var2: true

var2 <= var3: false

Logical Operators

Logical Operators

Logical operators act upon boolean operands only.

The outcome is always a value of type boolean.

In particular, “and” and “or” logical operators occur in two forms:

1) full op1 & op2 and op1 | op2 where both op1 and op2 are evaluated

2) short-circuit - op1 && op2 and op1 || op2 where op2 is only evaluated if the value of op1 is insufficient to determine the final outcome

Table: Logical Operators

&	Op1 & op2	AND
|	Op1 | op2	OR
&&	Op1 && op2	Short-circuit AND
||	Op1 || op2	Short-circuit OR
!	! op	NOT
^	Op1 ^ op2	XOR

Logical ‘AND‘ operator

Logical ‘AND’ Operator (&&) This operator returns true when both the conditions under consideration are satisfied or are true. If even one of the two yields false, the operator results false. In Simple terms, ***cond1 && cond2 returns true when both cond1 and cond2 are true (i.e. non-zero***).

For example:

```
class Logical {
  public static void main(String[] args)
  {
    // initializing variables
    int a = 10, b = 20, c = 20, d = 0;
    // Displaying a, b, c
    System.out.println("Var1 = " + a);
    System.out.println("Var2 = " + b);
    System.out.println("Var3 = " + c);
    // using logical AND to verify
    // two constraints
    if ((a < b) && (b == c)) {
      d = a + b + c;
      System.out.println("The sum is: " + d);
    }
    else
      System.out.println("False conditions");   }}
```

Output:

Var1 = 10

Var2 = 20

Var3 = 20

The sum is: 50

Logical ‘OR‘ operator

Logical ‘OR’ Operator (||) This operator returns true when one of the two conditions under consideration is satisfied or is true. If even one of the two yields true, the operator results true. To make the result false, both the constraints need to return false.

For example:

```
class Logical {
  public static void main(String[] args)
  {
    // initializing variables
    int a = 10, b = 1, c = 10, d = 30;
    // Displaying a, b, c
    System.out.println("Var1 = " + a);
    System.out.println("Var2 = " + b);
    System.out.println("Var3 = " + c);
    System.out.println("Var4 = " + d);
    // using logical OR to verify
    // two constraints
    if (a > b || c == d)
      System.out.println(
        "One or both + the conditions are true");
    else
      System.out.println(
        "Both the + conditions are false"); }}
```

Output:

Var1 = 10
Var2 = 1
Var3 = 10
Var4 = 30
One or both + the conditions are true

Logical ‘NOT‘ operator

Logical ‘NOT’ Operator (!) Unlike the previous two, this is a unary operator and returns true when the condition under consideration is not satisfied or is a false condition. Basically, if the condition is false, the operation returns true and when the condition is true, the operation returns false.

For example:

```
class Logical {
  public static void main(String[] args)
  {
    // initializing variables
    int a = 10, b = 1;

    // Displaying a, b, c
    System.out.println("Var1 = " + a);
    System.out.println("Var2 = " + b);

    // Using logical NOT operator
    System.out.println("!(a < b) = " + !(a < b));
    System.out.println("!(a > b) = " + !(a > b));
  }
}
```

Output:

Var1 = 10

Var2 = 1

!(a < b) = true

!(a > b) = false

Break 10 minutes

Expressions in Java

Type of Expressions in Java

An expression in Java is a series of operators, variables, and method calls constructed according to the syntax o the language for evaluating a single value.

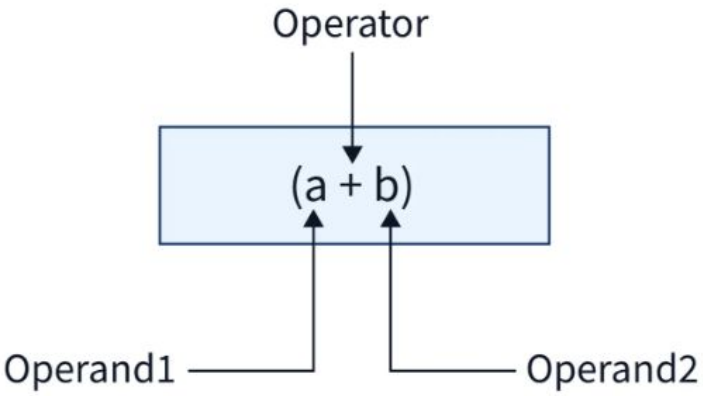
A simple expression in java has a type that can either be a primitive type or a reference type.

There are 3 types of expressions in Java:

- Infix Expression
- Postfix Expression
- Prefix Expression

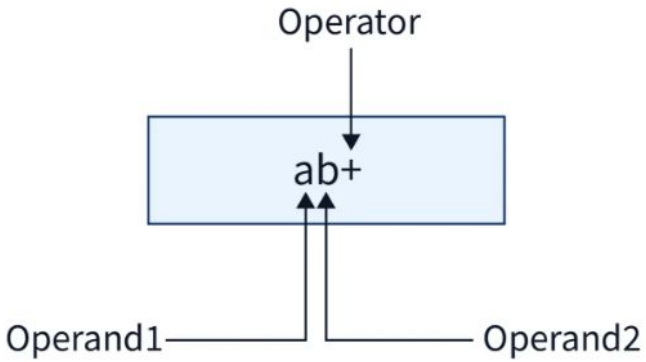
Infix Expression

It is an expression in which the operators are used between operands.



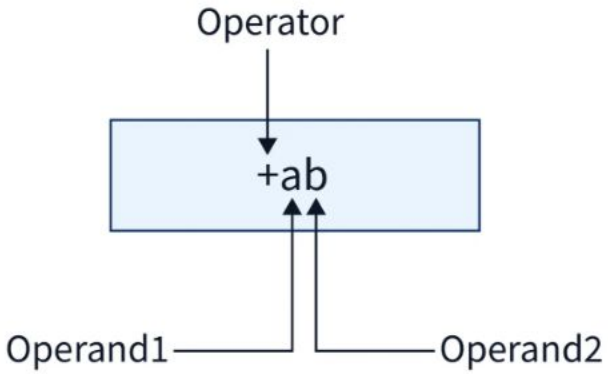
Postfix Expression

It is an expression in which operators are used after operands.



Prefix Expression

It is an expression in which operators are used before an operand.



Questions?

Thank you for your attention!