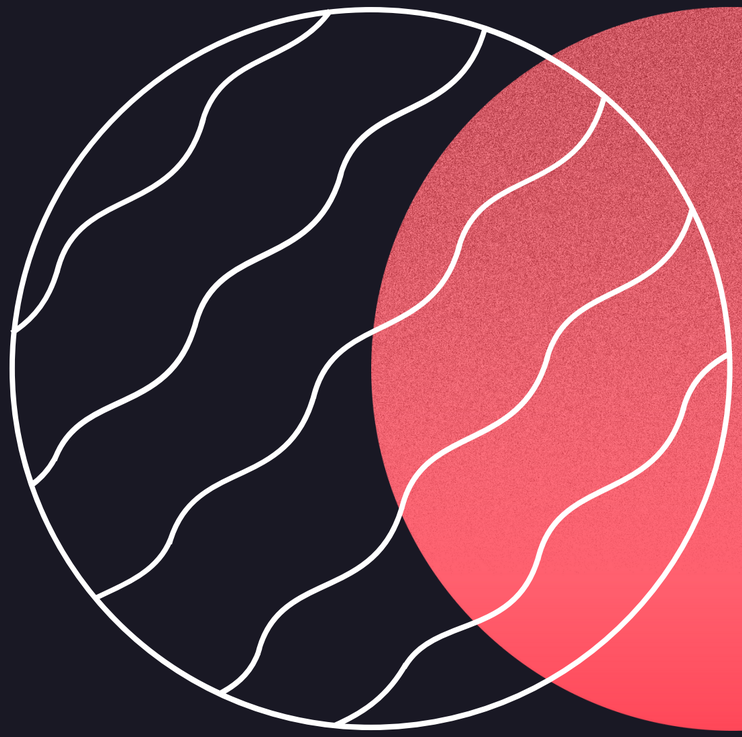
Google Developer Student Club
Meerut Institute of Engineering and Technology

DSA BOOTCAMP

SESSION 1
11FEB2023

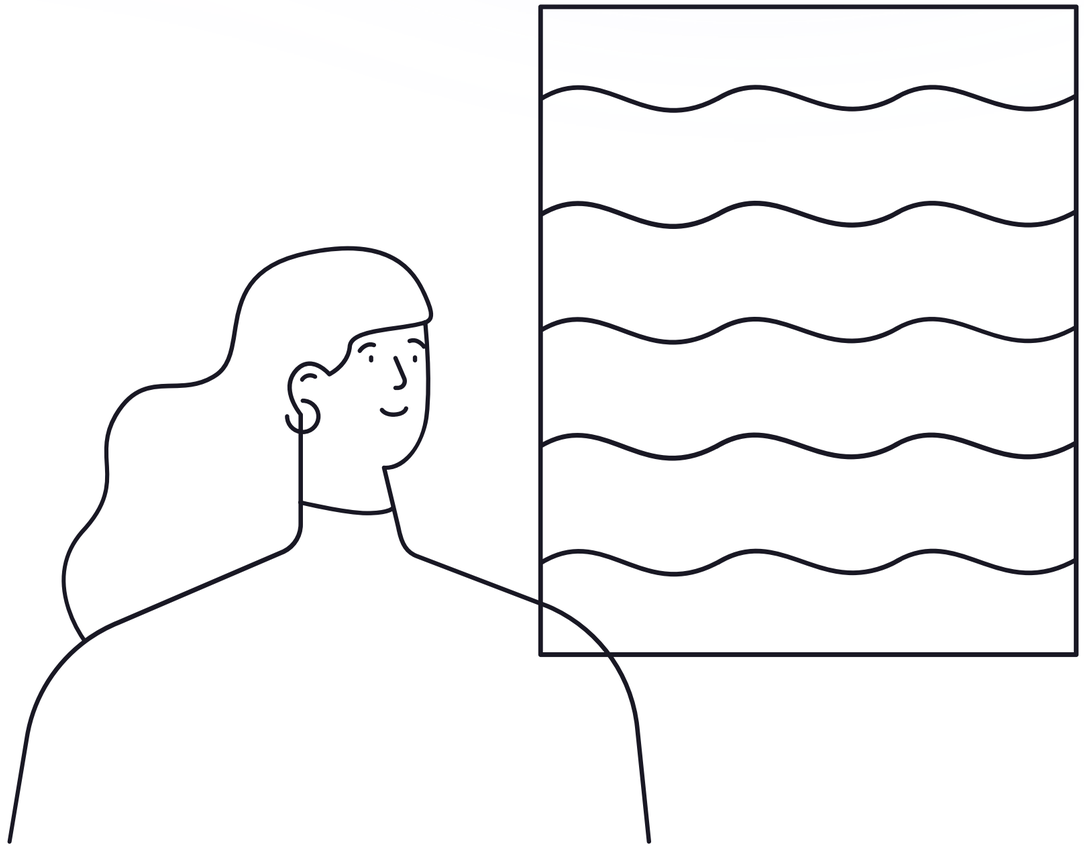


OPERATORS

OPERATORS ARE SYMBOLS THAT PERFORM OPERATIONS ON VARIABLES AND VALUES.

Types of Operators :

1. Arithmetic Operators
2. Assignment Operators
3. Relational Operator
4. Logical Operator
5. Bitwise Operator
6. Other Operators



Arithmetic Operators

- Perform arithmetic operations on variables and data

Operators	Operation
+	Additon
-	Subtraction
*	Multiplication
/	Division
%	Modulo operation

Lets take a = 7 and b=2,

Addition(+):

```
cout << "a + b = " << (a + b) << endl;
```

>>a + b = 9

Subtraction(-):

```
cout << "a - b = " << (a - b) << endl;
```

>>a - b = 5

Multiplication(*):

```
cout << "a * b = " << (a * b) << endl;
```

>>a * b = 14

Division(/):

```
cout << "a / b = " << (a / b) << endl;
```

>>a / b = 3

Modulo(%):

```
cout << "a % b = " << (a % b) << endl;
```

>>a % b = 1

INCREMENT AND DECREMENT OPERATORS

++ Operator

- Increases the value of the Operand by 1

-- Operator

- Decreases the value of the Operand by 1

Pre-increment (++i) – Before assigning the value to the variable, the value is incremented by one.

Post-increment (i++) – After assigning the value to the variable, the value is incremented.

SYNTAX:

```
++variable_name; // Pre-increment
variable_name++; // Post-increment
```

Pre-decrement (--i) – Before assigning the value to the variable, the value is decremented by one.

Post-decrement (i--) – After assigning the value to the variable, the value is decremented.

SYNTAX:

```
--variable_name; // Pre-decrement
variable_name--; // Post-decrement
```

WHAT IS THE OUTPUT OF THE FOLLOWING

```
int x = 5, y = 5, z;
x = ++x; y = --y;
z = x++ + y--;
cout << z;
```

WHAT IS THE OUTPUT OF THE FOLLOWING?

```
int x = 5, y = 5, z;      //x=5, y=5, z=...
x = ++x; y = --y;         //x=6, y=4, z=...
z = x++ + y--;            //x=6, y=4, z=10
cout << z;                //x=7, y=3, z=10
z= x+y;                   //x=7, y=3, z=10
```

Assignment Operators

- Used to assign values to variables

Operators	Example	Equivalent to
=	a = b	a = b
+=	a += b	a = a + b
-=	a -= b	a = a - b
*=	a *= b	a = a * b
/=	a /= b	a = a / b
%=	a %= b	a = a % b

Relational Operators

- tests or defines some kind of relation between two operands.

Operators	Meaning	Example
==	is equal to	3 == 5 gives us false
!=	Not equal to	3 != 5 gives us True
>	Greater than	3 > 5 gives us false
<	Less than	3 < 5 gives us True
>=	Greater than equal to	3 >= 5 gives us false
<=	Less than equal to	3 <= 5 gives us false

Bitwise Operators

- Used to perform operations on Individual bits
- They can only be used alongside char and int data types.

Operators	Description
&	Binary AND
|	Binary OR
^	Binary XOR
~	Binary One's Complement
>>	Binary shift right
<<	Binary shift left

X	Y	X&Y (AND)	X|Y (OR)	X^Y (XOR)	X~	Y~
0	0	0	0	0	1	1
0	1	0	1	1	1	0
1	0	0	1	1	0	1
1	1	1	1	0	0	0

Left shift operator(<<)

```
int a = 5;
//a = 0101
a << 2;
//0101<<2 =0001 0100
a=20
```

Right shift operator(>>)

```
int a = 7;
//a = 0111
a >> 2;
//0111>>2 = 0000 0001
a=1
```

Logical Operators

- used to check whether an expression is true or false.
- If the expression is true, it returns 1 whereas if the expression is false, it returns 0

Operators	Example	Meaning
&&	expression1 && expression2	Logical AND True only if all the operands are true
||	expression1 || expression2	Logical OR True only if at least one of the operands is true.
!	!expression	Logical NOT True only if the operand is false

Ternary Operator

A ternary operator evaluates the test condition and executes a block of code based on the result of the condition.

```
condition ? expression1 : expression2;
```

Here, ***condition*** is evaluated and

- if ***condition*** is true, ***expression1*** is executed.
- And, if ***condition*** is false, ***expression2*** is executed.