21 May

# Python Basic - 2

***Q.1. Create two int type variables, apply addition, subtraction, division and multiplications and store the results in variables. Then print the data in the following format by calling the variables:***

***First variable is __ & second variable is __.***

***Addition: __ + __ = __***

***Subtraction: __ - __ = __***

***Multiplication: __ * __ = __***

***Division: __ / __ = __***

***Ans:***

```
num1 = 10
num2 = 5
print("First variable is " , num1 , " & Second variable is ", num2)

add = num1 + num2
print("Addition : ", num1 ," + " , num2 , " = " ,add)

sub = num1 - num2
print("Subtraction  : ", num1 ," - " , num2 ," = " ,sub)

mul = num1 * num2
print("Multiplication : ", num1 ," * " , num2 ," = " , mul)

div = num1 / num2
print("Division : ", num1 ," / " , num2 ," = ",div)
```

***Output:-***

```
First variable is  10  & Second variable is  5
Addition :  10  +  5  =  15
Subtraction  :  10  -  5  =  5
Multiplication :  10  *  5  =  50
Division :  10  /  5  =  2.0
```

***Q.2. What is the difference between the following operators:***

(i) ***'/' & '//'***

(ii) ***'**' & '^'***

***Ans:-***

(i) ***'/' & '//'***

Both are division operators, but they have different behaviors.

- The **'/'** operator performs regular division and returns the quotient as a floating-point or decimal number. For example, 10 / 3 would give output as 3.3333333.

- The **'//'** operator returns the quotient as an integer by removing any decimal places. It rounds the result towards negative infinity. For example, 10 // 3 would be evaluated as 3, removing the decimal part.

(ii) ***'**' & '^'***

Both are used for exponentiation, but their usage differs:

- The **'**'** operator is called as exponentiation operator.It raises a base number to the power of an exponent. For example, 2 ** 3 would give output as 8 because it calculates 2 raised to the power of 3.

- The '**^'** operator is called as a bitwise XOR operator. It performs an operation on the binary representation of two numbers. For example, 5 ^ 3 will give output as 6 because the binary representation of 5 is 101 and 3 is 011. Performing XOR on these values results in 110, which is 6 in decimal representation.

Q.3. ***List the logical operators.***

***Ans:-***

1. **AND (&&):** The AND operator returns true if both operands are true, otherwise it returns false.
2. **OR (||):** The OR operator returns true if at least one of the operands is true, otherwise it returns false.
3. **NOT (!):** The NOT operator neglects the value of a Boolean expression. It returns true if the expression is false, and vice versa.

Q.4. ***Explain right shift operator and left shift operator with examples.***

***Ans:-***

- ***Right shift operator:-***

The right shift operator (>>) shifts the bits of a number to the right by a specified number of positions. It effectively divides the original number by 2 raised to the power of the shift amount. Here's an example:

```
x = 10  # Binary representation: 1010
y = x >> 2  # Shift right by 2 positions

# The binary representation of y after shifting right is 0010,
# which is equivalent to the decimal value 2.
print(y)  # Output: 2
```

- ***Left shift operator:-***

The left shift operator (<<) shifts the bits of a number to the left by a specified number of positions. It effectively multiplies the original number by 2 raised to the power of the shift amount. Here's an example:

```
x = 5  # Binary representation: 0101
y = x << 3  # Shift left by 3 positions

# The binary representation of y after shifting left is 0101000,
# which is equivalent to the decimal value 40.
print(y)  # Output: 40
```

Q.5. ***Create a list containing int type data of length 15. Then write a code to check if 10 is present in the list or not.***

***Ans:-***

```
my_list = [2, 5, 8, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65]

# Check if 10 is present in the list
if 10 in my_list:
    print("10 is present in the list.")
else:
    print("10 is not present in the list.")
```

***Output:***

```
10 is present in the list.
```