**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: \_\_ / \_\_ = \_\_**

Input: # Create two int type variables

first\_variable = 10

second\_variable = 5

# Perform arithmetic operations

addition = first\_variable + second\_variable

subtraction = first\_variable - second\_variable

multiplication = first\_variable \* second\_variable

division = first\_variable / second\_variable

# Print the results

print("First variable is", first\_variable, "& second variable is", second\_variable)

print("Addition:", first\_variable, "+", second\_variable, "=", addition)

print("Subtraction:", first\_variable, "-", second\_variable, "=", subtraction)

print("Multiplication:", first\_variable, "\*", second\_variable, "=", multiplication)

print("Division:", first\_variable, "/", second\_variable, "=", division)

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

In the code above, two int variables (first\_variable and second\_variable) are created with values 10 and 5 respectively. The arithmetic operations (addition, subtraction, multiplication, and division) are performed and stored in separate variables (addition, subtraction, multiplication, and division). Finally, the results are printed using the print() function with the desired format by calling the variables.

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

**(i) ‘/’ & ‘//’**

**(ii) ‘\*\*’ & ‘^’**

(i) Difference between '/' and '//':

The '/' operator is the division operator in Python and performs normal division, returning the result as a floating-point number.

The '//' operator is the floor division operator in Python. It performs division and returns the integer quotient by discarding the fractional part. It essentially performs division and rounds the result down to the nearest integer.

a = 10

b = 3

result1 = a / b # Normal division

result2 = a // b # Floor division

print(result1) # Output: 3.3333333333333335

print(result2) # Output: 3

In the example above, result1 is calculated using the '/' operator, which performs normal division and returns a floating-point result (3.3333333333333335). On the other hand, result2 is calculated using the '//' operator, which performs floor division and returns the integer quotient (3) by discarding the fractional part.

(ii) Difference between '\*\*' and '^':

The '\*\*' operator is the exponentiation operator in Python and is used to raise a number to a power. It returns the result of raising the base to the power of the exponent.

The '^' operator, on the other hand, is not an exponentiation operator in Python. It is used as the bitwise XOR operator for performing bitwise XOR operations on binary representations of numbers.

a = 2

b = 3

result1 = a \*\* b # Exponentiation

result2 = a ^ b # Bitwise XOR

print(result1) # Output: 8

print(result2) # Output: 1

In the example above, result1 is calculated using the '\*\*' operator, which raises 2 to the power of 3, resulting in 8. result2 is calculated using the '^' operator, which performs bitwise XOR on the binary representations of 2 and 3, resulting in 1.

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

In Python, the logical operators are used to perform logical operations on boolean values (True or False). Here are the logical operators in Python:

Logical AND (and): The logical AND operator returns True if both operands are True, otherwise it returns False.

Logical OR (or): The logical OR operator returns True if at least one of the operands is True, otherwise it returns False.

Logical NOT (not): The logical NOT operator returns the opposite of the operand. If the operand is True, it returns False, and if the operand is False, it returns True.

Here's a table summarizing the logical operators:

Operator Description

and Returns True if both operands are True

or Returns True if at least one operand is True

not Returns the opposite of the operand

These logical operators are often used in conditional statements and boolean expressions to make decisions and perform logical evaluations.

**Q.4. Explain right shift operator and left shift operator with examples.**In Python, the right shift (>>) and left shift (<<) operators are bitwise operators that perform bit-level operations on integers. These operators shift the binary representation of a number to the right or left by a specified number of positions.

Right Shift (>>) Operator:

The right shift operator shifts the binary representation of a number to the right by a specified number of positions. It discards the rightmost bits and introduces zeros on the left. The syntax is x >> y, where x is the number to be shifted, and y is the number of positions to shift.

x = 10 # Binary representation: 1010

y = 2

result = x >> y # Right shifting 2 positions

print(result) # Output: 2

In the example above, the binary representation of x is 1010. When we right-shift x by 2 positions using x >> y, we get the result 2, which corresponds to the binary representation 10. The rightmost bits are discarded, and zeros are introduced on the left.

Left Shift (<<) Operator:

The left shift operator shifts the binary representation of a number to the left by a specified number of positions. It introduces zeros on the right and shifts the bits to the left. The syntax is x << y, where x is the number to be shifted, and y is the number of positions to shift.

x = 10 # Binary representation: 1010

y = 2

result = x << y # Left shifting 2 positions

print(result) # Output: 40

In the example above, the binary representation of x is 1010. When we left-shift x by 2 positions using x << y, we get the result 40, which corresponds to the binary representation 101000. Zeros are introduced on the right, and the bits are shifted to the left.

**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**.

Input: # Create a list of integers

my\_list = [5, 8, 3, 12, 10, 6, 15, 9, 11, 2, 7, 4, 1, 13, 14]

# 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.

In the code above, we create a list named my\_list containing integers of length 15. We then use the in operator to check if 10 is present in the list. If it is, the code prints "10 is present in the list." If it is not, the code prints "10 is not present in the list."