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

# Creating two int type variables

num1 = 20

num2 = 10

# Performing arithmetic operations and storing results in variables

addition\_result = num1 + num2

subtraction\_result = num1 - num2

multiplication\_result = num1 \* num2

division\_result = num1 / num2

# Printing the results

print(f"First variable is {num1} & second variable is {num2}.")

print(f"Addition: {num1} + {num2} = {addition\_result}")

print(f"Subtraction: {num1} - {num2} = {subtraction\_result}")

print(f"Multiplication: {num1} \* {num2} = {multiplication\_result}")

print(f"Division: {num1} / {num2} = {division\_result}")

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

(i) ‘/’ & ‘//’

(ii) ‘\*\*’ & ‘^’

(i) `/` & `//`:

- `/` is the division operator in Python, which returns the floating-point result (float) of the division.

- `//` is the floor division operator in Python, which returns the integer result (int) of the division, rounded down to the nearest whole number (floor value).

# Division

result1 = 10 / 3

print(result1)

# Output: 3.3333333333333335

# Floor Division

result2 = 10 // 3

print(result2)

# Output: 3

(ii) `\*\*` & `^`:

- `\*\*` is the exponentiation operator in Python, which raises the left operand to the power of the right operand.

- `^` is not the exponentiation operator in Python. It is the bitwise XOR operator.

# Exponentiation

result3 = 2 \*\* 3

print(result3)

# Output: 8

# Bitwise XOR (Note: ^ is not an exponentiation operator in Python)

result4 = 2 ^ 3

print(result4)

# Output: 1

Q.3. List the logical operators.

The logical operators in Python are:

- `and`: Returns True if both operands are True.

- `or`: Returns True if at least one of the operands is True.

- `not`: Returns the opposite boolean value of the operand. If the operand is True, it returns False, and vice versa.

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

- Right Shift Operator (`>>`): The right shift operator shifts the bits of a number to the right by a specified number of positions. It discards the bits shifted off to the right and fills the leftmost positions with zeroes.

num = 12

result\_right\_shift = num >> 2

print(result\_right\_shift) # Output: 3

In this example, the binary representation of 12 is `1100`. When we right shift it by 2 positions, the result is `11`, which is equal to 3 in decimal.

- Left Shift Operator (`<<`): The left shift operator shifts the bits of a number to the left by a specified number of positions. It fills the rightmost positions with zeroes.

num = 3

result\_left\_shift = num << 2

print(result\_left\_shift) # Output: 12

```

In this example, the binary representation of 3 is `11`. When we left shift it by 2 positions, the result is `1100`, which is equal to 12 in decimal.

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.

# Creating a list containing int type data of length 15

my\_list = [1, 3, 5, 7, 9, 10, 12, 15, 17, 20, 22, 25, 30, 35, 40]

# Checking 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.

```