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

Here's the code that creates two `int` variables, performs arithmetic operations, and stores the results in variables. It then prints the data in the requested format:

# Create 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)

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

Note that in Python, the division operation `/` returns a float, even if the result is a whole number.

Q.2. What is the difference between the following operators:   
(i) ‘/’ & ‘//’  
(ii) ‘\*\*’ & ‘^’

The operators you mentioned have different functionalities:

(i) '/': This is the division operator in Python. It performs floating-point division, which means that it returns a float as the result. For example, 5 / 2 would give 2.5.

(ii) '//': This is the floor division operator in Python. It performs division and rounds down the result to the nearest whole number. It returns an integer as the result. For example, 5 // 2 would give 2.

Q.3. List the logical operators.

The logical operators in Python are:

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. If both operands are false, it returns `False`.

3. `not`: The `not` operator is a unary operator that negates the value of its operand. It returns `True` if the operand is false, and `False` if the operand is true.

These logical operators are used to evaluate conditions and perform logical operations in Python.

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

The right shift (`>>`) and left shift (`<<`) operators are bitwise shift operators in Python. They manipulate the binary representations of integers by shifting the bits to the right or left, respectively. Here's an explanation of each operator with examples:

1. Right Shift (`>>`): The right shift operator (`>>`) shifts the bits of a number to the right by a specified number of positions. It effectively divides the number by 2 for each shift to the right, discarding the least significant bit and shifting in zeros from the left.

Example:

number = 10 # Binary representation: 1010

shifted = number >> 2

print(shifted) # Output: 2

In this example, the binary representation of `10` is `1010`. Shifting the bits two positions to the right results in `10`, which is the decimal representation of `2`.

2. Left Shift (`<<`):

The left shift operator (`<<`) shifts the bits of a number to the left by a specified number of positions. It effectively multiplies the number by 2 for each shift to the left, filling in zeros from the right.

Example:

number = 5 # Binary representation: 101

shifted = number << 3

print(shifted) # Output: 40

In this example, the binary representation of `5` is `101`. Shifting the bits three positions to the left results in `101000`, which is the decimal representation of `40`.

The bitwise shift operators (`>>` and `<<`) are typically used in low-level programming and manipulating binary representations of numbers for various purposes, such as optimizing code or working with flags and bit patterns.

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.

Here's an example code that creates a list of length 15 containing integer values and checks if 10 is present in the list:

# Create a list of integers

my\_list = [5, 2, 8, 10, 3, 15, 7, 12, 1, 4, 6, 9, 11, 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 this code, we create a list `my\_list` with 15 integer values. We then use the `in` operator to check if the value `10` is present in the list. If it is, we print the message "10 is present in the list." Otherwise, we print the message "10 is not present in the list."