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: __ / __ = __

Answer:-
here's the code to create two integer variables, perform addition, subtraction, multiplication, and division, and then print the results in the requested format:

```
a= 10
b= 5
addition_result = a+ b
subtraction_result = a- b
multiplication_result = a* b
division_result = a/ b

# Print the data in the specified format
print(f"First variable is {a} & second variable is {b}.")
print(f"Addition: {a} + {b} = {addition_result}")
print(f"Subtraction: {a} - {b} = {subtraction_result}")
print(f"Multiplication: {a} * {b} = {multiplication_result}")
print(f"Division: {a} / {b} = {division_result}")
```

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) '**' & '^'

Answer:-
(i) /' & '//':

/ is the division operator in Python, used to perform regular division between two operands, and it returns a floating-point result.
// is the floor division operator in Python, used to perform integer division between two operands, and it returns the largest integer that is less than or equal to the division result.

In the first case, 10 / 3 results in 3.3333333333333335 (floating-point division), while in the second case, 10 // 3 results in 3 (integer division).

(ii) ** & ^:

** is the exponentiation operator in Python, used to raise a number to a power.
^ is not an exponentiation operator in Python, but rather it is the bitwise XOR operator, used for bitwise exclusive OR operation on the binary representations of the operands.

In the first case, 2 ** 3 results in 8 (2 raised to the power of 3), while in the second case, 5 ^ 3 results in 6 (bitwise XOR of binary 101 and 011).

Q.3. List the logical operators.

Answer:-
the logical operators are used to perform logical operations on boolean values (True or False). There are three logical operators in Python:

- and: The logical AND operator returns True if both operands are True, otherwise, it returns False.
- or: The logical OR operator returns True if at least one of the operands is True, otherwise, it returns False.
- not: The logical NOT operator returns the negation of the operand. If the operand is True, not will return False, and if the operand is False, not will return True.

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

Answer:-
In Python, the right shift operator (>>) and left shift operator (<<) are bitwise shift operators. They are used to shift the binary representation of integers to the right or left by a specified number of positions. These operators work at the bit level, shifting the bits of the integer representation.

Right Shift Operator (>>):
The right shift operator shifts the bits of an integer to the right by the specified number of positions. For each right shift by one position, the integer is divided by 2 (equivalent to integer division). The rightmost bits are discarded, and the leftmost bits are filled with the sign bit (the most significant bit, which indicates the sign of the number). For positive numbers, this will be a zero, and for negative numbers, this will be a one.

Example:
```
# Right shift by 1 position
number = 8
result = number >> 1  # 8 >> 1 is equivalent to 8 // 2
print(result)  # Output: 4
```
In this example, the binary representation of 8 is 0b1000. After right-shifting by one position, the result is 0b0100, which is equivalent to 4.

Left Shift Operator (<<):
The left shift operator shifts the bits of an integer to the left by the specified number of positions. For each left shift by one position, the integer is multiplied by 2. The rightmost bits are filled with zeros,

and the leftmost bits are discarded. Left shifting can result in overflow if the shifted bits go beyond the number of bits used to represent integers in Python.

Example:

```
# Left shift by 2 positions
number = 3
result = number << 2  # 3 << 2 is equivalent to 3 * 2 * 2
print(result)  # Output: 12
```

In this example, the binary representation of 3 is 0b11. After left-shifting by two positions, the result is 0b1100, which is equivalent to 12.

Bitwise shift operators are often used in low-level programming and to perform certain optimizations. However, their usage is relatively less common in high-level Python programming compared to other operations.

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.

Answer:-

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

# 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.")
```