***ASSIGNMENT\_7***

**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:**

Here's a code snippet that creates two integer variables, performs addition, subtraction, multiplication, and division, and prints the results in the desired format:

# Create two integer variables

num1 = 10

num2 = 5

# Perform arithmetic operations and store the results in variables

addition = num1 + num2

subtraction = num1 - num2

multiplication = num1 \* num2

division = num1 / num2

# Print the data in the desired format

print("First variable is", num1, "& second variable is", num2)

print("Addition: {} + {} = {}".format(num1, num2, addition))

print("Subtraction: {} - {} = {}".format(num1, num2, subtraction))

print("Multiplication: {} \* {} = {}".format(num1, num2, multiplication))

print("Division: {} / {} = {}".format(num1, num2, 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

Explanation:

Two integer variables, num1 and num2, are created with values 10 and 5, respectively.

Addition, subtraction, multiplication, and division operations are performed using the +, -, \*, and / operators, respectively, and the results are stored in the corresponding variables: addition, subtraction, multiplication, and division.

The print() function is used to display the output in the desired format. The variables num1, num2, and the arithmetic operation results are inserted into the string using the {} placeholders, and formatted using the .format() method.

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

**(i) ‘/’ & ‘//’**

**(ii) ‘\*\*’ & ‘^’**

***ANS:***

The operators you mentioned have different functionalities and purposes in Python:

**(i) / and // operators:**

The / operator performs floating-point division, which means it returns the result as a float value, even if the operands are integers. For example, 10 / 3 would result in 3.3333333333333335.

The // operator performs floor division or integer division. It returns the largest integer that is less than or equal to the division result. It effectively truncates any decimal portion and returns an integer. For example, 10 // 3 would result in 3.

**Example:**

**print(10 / 3) # Output: 3.3333333333333335**

**print(10 // 3) # Output: 3**

**(ii) \*\* and ^ operators:**

The \*\* operator is used for exponentiation. It raises the left operand to the power of the right operand. For example, 2 \*\* 3 would result in 8.

The ^ operator, on the other hand, is the bitwise XOR operator in Python. It performs the bitwise XOR operation on the binary representations of the operands. It does not represent exponentiation as it does in some other programming languages.

**Example:**

print(2 \*\* 3) # Output: 8

print(2 ^ 3) # Output: 1 (bitwise XOR of binary 10 and 11)

To summarize:

The / operator performs floating-point division.

The // operator performs floor division, returning an integer result.

The \*\* operator represents exponentiation.

The ^ operator represents bitwise XOR operation.

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

***ANS:***

In Python, the logical operators are used to perform logical operations on boolean values or expressions. Here are the three logical operators in Python:

**and Operator:**

The and operator returns True if both operands are True, otherwise, it returns False. It evaluates the second operand only if the first operand is True. If the first operand is False, the second operand is not evaluated.

Example:

x = 5

y = 10

z = 15

print(x < y and y < z) # Output: True

print(x > y and y < z) # Output: False

**or Operator:**

The or operator returns True if at least one of the operands is True, otherwise, it returns False. It evaluates the second operand only if the first operand is False. If the first operand is True, the second operand is not evaluated.

Example:

x = 5

y = 10

z = 15

print(x < y or y < z) # Output: True

print(x > y or y > z) # Output: False

**not Operator:**

The not operator is a unary operator that returns the negation of the operand. It returns True if the operand is False, and False if the operand is True.

Example:

x = True

y = False

print(not x) # Output: False

print(not y) # Output: True

These logical operators can be used to combine conditions and perform logical operations in control flow statements and conditional expressions.

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

***ANS****:*

In Python, the right shift (>>) and left shift (<<) operators are used for bitwise shifting of the binary representations of integers. Here's an explanation of these operators with examples:

Right Shift Operator (>>):

The right shift operator shifts the bits of the left operand to the right by the number of positions specified by the right operand. It effectively divides the left operand by 2 raised to the power of the right operand.

Syntax: operand >> shift\_count

Example:

x = 10 # Binary: 1010

# Right shifting x by 2 positions

result = x >> 2 # Binary: 10

print(result) # Output: 2

In the example, the binary representation of x is 1010. Shifting the bits to the right by 2 positions results in 10, which is equivalent to the decimal value of 2.

Left Shift Operator (<<):

The left shift operator shifts the bits of the left operand to the left by the number of positions specified by the right operand. It effectively multiplies the left operand by 2 raised to the power of the right operand.

Syntax: operand << shift\_count

Example:

x = 5 # Binary: 101

# Left shifting x by 2 positions

result = x << 2 # Binary: 10100

print(result) # Output: 20

In the example, the binary representation of x is 101. Shifting the bits to the left by 2 positions results in 10100, which is equivalent to the decimal value of 20.

The right shift and left shift operators are useful for performing bitwise operations, such as manipulating individual bits or optimizing certain operations involving binary representations.

**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:***

Certainly! Here's a code snippet that creates a list of integers of length 15 and checks if the number 10 is present in the list:

# Create a list of integers

my\_list = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 10, 27]

# 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.

Explanation:

The code creates a list called my\_list containing 15 integer values.

The in operator is used to check if the value 10 is present in the list.

If 10 is found in the list, the code prints the message "10 is present in the list."

If 10 is not found in the list, the code prints the message "10 is not present in the list."

By using the in operator, you can easily check the presence of a specific value in a list or any other iterable object in Python.