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:___/___= ___

Ans: first_variable = 10
second_variable = 5

```
# Perform arithmetic operations and store results in variables
addition_result = first_variable + second_variable
subtraction_result = first_variable - second_variable
multiplication_result = first_variable * second_variable
division_result = first_variable / second_variable

# Print the results in the desired format
print("First variable is\t\t", first_variable, "\t& second variable is\t", second_variable)
print("Addition:\t\t", "+", "\t=\t", addition_result)
print("Subtraction:\t\t", "-", "\t=\t", subtraction_result)
print("Multiplication:\t\t", "*", "\t=\t", multiplication_result)
print("Division:\t\t", "/", "\t=\t", division_result)
```

Output:

vbnet

```
First variable is        10      & second variable is     5
Addition:               +       =       15
Subtraction:             -       =       5
Multiplication:          *       =       50
Division:               /       =       2.0
```

In this code, we create two int variables first_variable and second_variable with values 10 and 5, respectively. We then perform addition, subtraction, multiplication, and division operations using these variables and store the results in separate variables.

Finally, we use print() statements to display the values in the desired format, using string concatenation and separating the components with tabs (\t). The variables and arithmetic operators are included within the print statements to present the results in the specified format.

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

(i) '/' & '//'

(ii) '**' & '^'

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

The '/' operator performs division and returns a floating-point result.

The '//' operator performs floor division and returns the quotient as an integer, discarding the decimal part.

Here's an example to illustrate the difference:

python

```
a = 10
b = 3

division_result = a / b
floor_division_result = a // b

print(division_result)          # Output: 3.3333333333333335
print(floor_division_result)    # Output: 3
```

In this example, a is divided by b using the '/' operator, resulting in a floating-point result of approximately 3.3333333333333335. On the other hand, the '//' operator is used for floor division, which discards the decimal part and returns the quotient as an integer, resulting in 3.

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

The '**' operator is used for exponentiation or raising a number to a power.

The '^' operator is not an exponentiation operator in Python. Instead, it is used for bitwise XOR (exclusive OR) operation.

Here's an example to illustrate the difference:

python

```
a = 2
b = 3

exponentiation_result = a ** b
bitwise_xor_result = a ^ b

print(exponentiation_result)    # Output: 8
print(bitwise_xor_result)       # Output: 1
```

In this example, a is raised to the power of b using the '**' operator, resulting in the value of 8. The '^' operator is used for bitwise XOR operation, which performs the XOR operation between the binary representations of a and b, resulting in 1.

Q.3. List the logical operators.

Ans: The logical operators in Python are as follows:

Logical AND (and): Returns True if both operands are True, otherwise returns False.

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

Logical NOT (not): Returns the negation of the operand. If the operand is True, it returns False. If the operand is False, it returns True.

These logical operators are used to perform logical operations on boolean values or expressions. They are typically used in conditional statements, boolean expressions, and control flow statements to evaluate and control the flow of the program based on certain conditions.

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

Ans: In Python, the right shift operator (>>) and the left shift operator (<<) are bitwise operators used to shift the binary representation of integers to the right or left, respectively. Here's an explanation of how these operators work, along 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 effectively divides the number by 2 for each shift to the right, discarding the least significant bit and shifting in zeros from the left.

The general syntax for the right shift operator is x >> y, where x is the number being shifted and y is the number of positions to shift.

Example:

python

```
number = 12  # Binary: 1100

# Right shifting by 2 positions
result = number >> 2  # Binary: 0011, Decimal: 3

print(result)  # Output: 3
```

In this example, the binary representation of the number 12 is 1100. When we perform a right shift by 2 positions (number >> 2), the bits are shifted to the right, resulting in 0011, which is the binary representation of the number 3. Therefore, the output is 3.

Left Shift Operator (<<):

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.

The general syntax for the left shift operator is x << y, where x is the number being shifted and y is the number of positions to shift.

Example:

python

```
number = 3  # Binary: 0011

# Left shifting by 2 positions
result = number << 2  # Binary: 1100, Decimal: 12

print(result)  # Output: 12
```

In this example, the binary representation of the number 3 is 0011. When we perform a left shift by 2 positions (number << 2), the bits are shifted to the left, resulting in 1100, which is the binary representation of the number 12. Therefore, the output is 12.

The right shift operator (>>) and the left shift operator (<<) are primarily used in bitwise operations and can be useful in scenarios that involve manipulating and extracting specific bits in binary representations of numbers

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 an example code that creates a list of length 15 containing integer type data, and then checks if the number 10 is present in the list:

python

```
# Create a list of integers
my_list = [5, 2, 8, 10, 3, 7, 6, 1, 9, 4, 12, 15, 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:

csharp

Copy code

```
10 is present in the list
```

In this code, we create a list my_list containing 15 integer elements. We then use the in operator to check if the number 10 is present in the list. If it is, we print the message "10 is present in the list". Otherwise, if 10 is not present in the list, we print the message "10 is not present in the list".

By running this code, you will get the output indicating whether the number 10 is present or not in the list.