

EXPERIMENT NO:01

Title: Write python program display welcome message on screen.

Prior Concepts: Python Installation and tools.

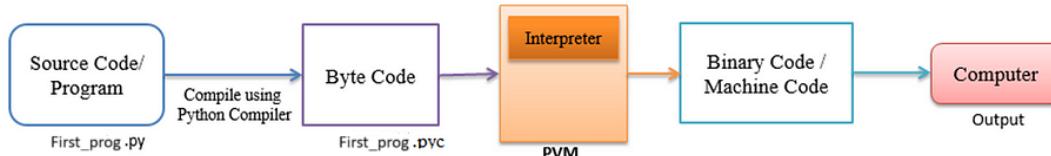
Objective:

- i. To get acquainted with the basics of Python programming.
- ii. To understand the usage of the print() function with its parameters.
- iii. To explore how to use comments in Python.

Theory:

Python is a high-level, interpreted programming language that enhances code readability and simplicity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python programs are also platform independent. Once we write a Python program, it can run on any platform without rewriting.

1. **Internal Working of Python:** A computer can understand only machine code that comprises 1s and 0s. So, it becomes obvious to convert source code into machine code for execution. For this a compiler is used. A compiler is a piece of program that normally converts the source code into machine code.



A Python compiler performs same task in a slightly different manner. It converts the source code into another code, called byte code. Python Virtual Machine (PVM) takes those byte codes instructions into machine code.

To carry out this conversion, PVM is equipped with an interpreter. The interpreter converts the byte code into machine code for further execution. The interpreter plays significant role hence Python Virtual Machine is also called an interpreter.

2. **Basics of Python Programming:** Python is a high-level, interpreted programming language that emphasizes code readability and simplicity. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

- i. **Variables:** Variables are used to store data. In Python, you do not need to declare the type of the variable; it is inferred from the value assigned.

```

x = 10
name = "This is a String Variable"
  
```

- ii. **Data Types:** Python supports various data types, such as integers (int), floating-point numbers (float), strings (str), lists (list), dictionaries (dict), and more.
 - iii. **Operators:** Python supports basic arithmetic operators (+, -, *, /), comparison operators (==, !=, >, <), and logical operators (and, or, not).
3. Comments in Python: Comments are non-executable lines in the code that are used to describe what the code does. They are essential for making the code more readable and maintainable.
- Single-line Comment:** Starts with a #.

```
# This is a single-line comment
```

Multi-line Comment: Can be created using triple quotes (" or """).

```
"""
This is a multi-line comment.
It spans multiple lines.
"""
```

4. **The print() Function:** The print() function in Python is used to output data to the console. It is one of the most commonly used functions.

Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

- objects: The values to be printed. These can be strings, variables, or any other data type.
- sep: Defines the separator between the objects. The default is a space (' ').
- end: Specifies what to print at the end. The default is a newline character ('\n').
- file: Determines where to send the output. The default is sys.stdout (the console).
- flush: Controls whether the output is flushed (forced to be written) or not. The default is False.

Stepwise Procedure:

Step 1: Setting Up the Environment

- Open your Python IDE (Integrated Development Environment) or a text editor.
- Create a new Python file (.py) where you will write your code.

Step 2: Writing Basic Python Code

```
# Example script to demonstrate the print() function with all parameters

# Importing the sys module to use sys.stdout
import sys
```

```
# 1. Basic usage with multiple objects
print("Hello", "World", "!", sep=" ", end="\n")

# 2. Using the 'sep' parameter to separate objects with a custom separator
print("Python", "is", "fun", sep=" - ")
# 3. Using the 'end' parameter to avoid the newline and print on the same
line
print("This is", end=" ")
print("still on the same line.")

# 4. Redirecting the output to a file using the 'file' parameter
# Open a file in write mode
with open("output.txt", "w") as f:
    print("This will be written to a file.", file=f)

# 5. Using the 'flush' parameter to force the output to be flushed
# Here, it won't make a visible difference, but is crucial in some real-
time scenarios
print("Flushing this immediately.", flush=True)

# Checking the output written to the file
with open("output.txt", "r") as f:
    content = f.read()
    print("\nContent of 'output.txt':")
    print(content)
```

Step 3: Running the Python Script

- Save your Python file.
- Run the script in your IDE or using the command line:
- `python your_script_name.py`
- Observe the outputs printed on the console as per the code written.

Conclusion: Thus, we have understood commenting and `print()` function in Python.

Questions:

1. Write a Python program that prints the following pattern using the `print()` function with the `sep` and `end` parameters:

```
*  
* *  
* * *  
* * * *
```

Hint: Use multiple print() statements and the end parameter.

2. Write a Python script that calculates the factorial of a number using a loop. Add comments explaining each step in the code.
3. String Manipulation with print(): Write a program that takes a string input from the user and prints it in reverse order, with each character separated by a comma. Example:
4. Write a Python script that calculates the sum of the first 10 natural numbers. Use multi-line comments to explain how the sum is derived mathematically, then write the code.

EXPERIMENT NO:02

Title: Implement a python program using following operators: Arithmetic, Relational, logical, Assignment, Bitwise, Membership, Identity.

Prior Concepts: Basic python concepts, printing and commenting.

Objective: To understand and implement the functionality of Arithmetic, Relational, Logical, Assignment, Bitwise, Membership, and Identity operators.

Theory:

- 1. Arithmetic Operators:** Arithmetic operators are used to perform basic mathematical operations.
 - Addition (+): Adds two operands.
 - Subtraction (-): Subtracts the second operand from the first.
 - Multiplication (*): Multiplies two operands.
 - Division (/): Divides the first operand by the second.
 - Modulus (%): Returns the remainder of a division.
 - Exponentiation (**): Raises the first operand to the power of the second.
 - Floor Division (//): Returns the integer part of a division.
- 2. Relational Operators:** Relational operators compare two values and return a Boolean result (True or False).
 - Equal (==): Checks if two operands are equal.
 - Not Equal (!=): Checks if two operands are not equal.
 - Greater Than (>): Checks if the first operand is greater than the second.
 - Less Than (<): Checks if the first operand is less than the second.
 - Greater Than or Equal To (>=): Checks if the first operand is greater than or equal to the second.
 - Less Than or Equal To (<=): Checks if the first operand is less than or equal to the second.
- 3. Logical Operators:** Logical operators are used to combine conditional statements.
 - Logical AND (and): Returns True if both operands are true.
 - Logical OR (or): Returns True if at least one operand is true.
 - Logical NOT (not): Reverses the result of the operand.
- 4. Assignment Operators:** Assignment operators are used to assign values to variables.
 - Simple Assignment (=): Assigns the right-hand operand to the left-hand operand.
 - Add and Assign (+=): Adds the right-hand operand to the left-hand operand and assigns the result to the left-hand operand.

- Subtract and Assign (-=): Subtracts the right-hand operand from the left-hand operand and assigns the result to the left-hand operand.
- Multiply and Assign (*=): Multiplies the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
- Divide and Assign (/=): Divides the left-hand operand by the right-hand operand and assigns the result to the left-hand operand.
- Modulus and Assign (%=): Takes the modulus of the left-hand operand with the right-hand operand and assigns the result to the left-hand operand.

5. Bitwise Operators: Bitwise operators work on binary digits (bits) of operands.

- AND (&): Performs a bitwise AND operation.
- OR (|): Performs a bitwise OR operation.
- XOR (^): Performs a bitwise XOR operation.
- NOT (~): Performs a bitwise NOT operation.
- Left Shift (<<): Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
- Right Shift (>>): Shifts the bits of the left operand to the right by the number of positions specified by the right operand.

6. Membership Operators: Membership operators test for membership in a sequence (like a string, list, or tuple).

- in: Returns True if the value is found in the sequence.
- not in: Returns True if the value is not found in the sequence.

7. Identity Operators: Identity operators compare the memory locations of two objects.

- is: Returns True if both operands refer to the same object.
- is not: Returns True if both operands do not refer to the same object.

Stepwise Procedure and Its Output

Step 1: Setting Up the Environment

- Open your Python IDE or a text editor.
- Create a new Python file (.py) where you will write the code for each type of operator.

Step 2: Implementing Python Programs for Each Operator

```
#Arithmetic Operators:
a = 15
b = 4
print("Addition:", a + b)
```

```
print("Subtraction:", a - b)
print("Multiplication:", a * b)
print("Division:", a / b)
print("Modulus:", a % b)
print("Exponentiation:", a ** b)
print("Floor Division:", a // b)
```

#Relational Operators:

```
x = 10
y = 20
print("x == y:", x == y)
print("x != y:", x != y)
print("x > y:", x > y)
print("x < y:", x < y)
print("x >= y:", x >= y)
print("x <= y:", x <= y)
```

#Logical Operators:

```
a = True
b = False
print("a and b:", a and b)
print("a or b:", a or b)
print("not a:", not a)
```

#Assignment Operators:

```
a = 10
a += 5
print("a += 5:", a)
a -= 3
print("a -= 3:", a)
a *= 2
print("a *= 2:", a)
a /= 4
print("a /= 4:", a)
a %= 3
print("a %= 3:", a)
```

#Bitwise Operators:

```
a = 5 # Binary: 0101
b = 3 # Binary: 0011
print("a & b:", a & b) # AND
```

```

print("a | b:", a & b) # OR
print("a ^ b:", a ^ b) # XOR
print("~a:", ~a) # NOT
print("a << 1:", a << 1) # Left Shift
print("a >> 1:", a >> 1) # Right Shift

#Membership Operators:
numbers = [1, 2, 3, 4, 5]
print("3 in numbers:", 3 in numbers)
print("7 not in numbers:", 7 not in numbers)

#Identity Operators:
x = [1, 2, 3]
y = x
z = [1, 2, 3]
print("x is y:", x is y)
print("x is not z:", x is not z)

```

Step 3: Running the Python Script

- Save your Python file.
- Run the script in your IDE or using the command line:
- `python your_script_name.py`
- Observe the outputs printed on the console as per the code written.

Conclusion: Thus, we have understood the use of different operators in python.

Questions:

1. Write a program that converts a decimal number into binary and then performs bitwise operations (AND, OR, XOR, NOT) on two binary numbers. Then, use logical operators to check conditions based on the results.
2. Implement a Python program that uses compound assignment operators to solve an equation step-by-step, such as calculating the area of a triangle given base and height, and then adding a constant to the result.
3. Write a Python program that tests for membership of an element in a nested list or tuple. For example, check if the number 7 is present in [(1, 2, 3), [4, 5, 6], {7, 8, 9}].
4. Create a Python program that compares two objects using identity operators. Modify one object and check how it affects the identity comparison.