THADOMAL SHAHANI ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY

Roll no: I-62

3.Control Structure: LO2

1)Aim:

Write a Python program to print a triangle pattern (give any), emphasizing the transition from C to Python syntax.

Theory:

- Patterns can be generated using control structures like loops and conditionals.
- Loops handle repetitive processes, while conditionals direct decisions.
- Python's straightforward syntax simplifies coding and minimizes complexity.
- Transitioning from C to Python requires grasping indentation rules and dynamic typing.
- The range() function helps manage loop iterations effectively.
- Nested loops are a standard approach for creating patterns.
- String formatting techniques aid in shaping the final output

Program:

```
x = int(input("Enter the number of rows:"))
print("Diamond")
for i in range(x):
    for j in range(x-i):
        print(" ",end=" ")
    for k in range(2*i-1):
        print("*",end=" ")
    print()
for i in range(x):
    for j in range(i):
        print(" ",end=" ")
    for k in range(2*(x-i)-1):
        print("*",end=" ")
    print()
print()
```

Output:

```
Enter the number of rows:5
Diamond
    *
   * * *
  * * * * *
 * * * * * * *
* * * * * * * * *
 * * * * * * *
  * * * * *
   * * *
    *
```

Conclusion:

Using loops and conditionals, we can easily create patterns, showing how Python makes coding simple and flexible.

2)Aim:

Develop a Python program that takes a numerical input and identifies whether it is even or odd, utilizing conditional statements and loops.

Theory:

- Conditional statements are used to verify conditions like divisibility.
- The modulus operator (%) helps determine whether a number is even or odd.
- Loops enable the evaluation of multiple numbers in a sequence.
- Decision-making is managed using the if-else construct.
- Input validation ensures users provide valid data, avoiding errors.
- The int() function converts user input into a numerical format for processing.

Program:

```
while True:
    x = int(input("Enter any number:"))
    if(x < 0 and x == 0):
        print("Invalid Input!")
        continue
    if x%2 == 0:
        print(f"The number {x} is even")
    else:
        print(f"The number {x} is odd")
    break
```

Output;

Enter any number:5
The number 5 is odd

Conclusion:

Python uses simple checks to sort numbers as even or odd with the help of conditional logic.

3)Aim:

Design a Python program to compute the factorial of a given integer N.

Theory:

- Factorial refers to the product of all positive integers from 1 to a given number.
- It can be calculated using loops or recursive methods.
- Python supports large integers, enabling factorial calculations for very large numbers.
- The math.factorial() function offers a quick, built-in way to compute factorials.
- Recursive methods make factorial computation logic more streamlined and concise.

Program:

```
def fact(n):
    if (n==1 or n ==0):
        return 1
    elif(n<0):
        print("Invalid Input")
    else:
        return n*fact(n-1)

a = int(input("Enter any number:"))
x = fact(a)
print(f"Factorial of {a} is {x}")
```

Output:

Enter any number:5
Factorial of 5 is 120

Conclusion:
Python can quickly calculate factorials using loops or recursion, making it useful for many tasks.

4)Aim:

Using function, write a Python program to analyze the input number is prime or not.

Theory:

- Prime numbers can only be divided evenly by 1 and themselves.
- Using functions helps break down the logic for checking primes into reusable pieces.
- A common method involves looping through numbers to test divisibility.
- The `sqrt()` function improves efficiency by reducing the range of checks.
- Advanced algorithms like the Sieve of Eratosthenes are effective for finding primes in large sets.

Program:

```
def prime_number(n):
    if n<= 1:
        return False
    for i in range(2,n):
        if n % i == 0:
            return False
        else:
            return True

x = int(input("Enter a number: "))

if prime_number(x):
    print(f"{x} is a prime number!")
else:
    print(f"{x} is not a prime number.")
```

Output:

```
Enter a number: 6
6 is not a prime number.
Enter a number: 5
5 is a prime number!
```

Conclusion:

Using functions to check if numbers are prime makes the code easier to organize and reuse.

5)Aim:

Implement a simple Python calculator that takes user input and performs basic arithmetic operations (addition, subtraction, multiplication, division) using functions.

Theory:

- Arithmetic tasks like addition, subtraction, multiplication, and division are managed using functions.
- Proper input handling and error checks ensure a smooth user experience.
- Modular code structure makes it simpler to add advanced operations later.
- The `eval()` function allows dynamic arithmetic calculations but poses security risks.
- Preventing division by zero is key to avoiding runtime issues.

Program:

```
def add(a, b):
   return a + b
def sub(a, b):
   return a - b
def multiply(a, b):
   return a * b
def divide(a, b):
   if b != 0:
      return a / b
   else:
      return "Error! Division by Zero gives Infinity"
num1 = float(input("Enter the 1st Number:"))
num2 = float(input("Enter the 2st Number:"))
print(f"The result of Addition is {add(num1, num2)}")
print(f"The result of Subtraction is {sub(num1, num2)}")
print(f"The result of Multiplication is {multiply(num1, num2)}")
print(f"The result of Division is {divide(num1, num2)}")
```

Output:

```
Enter the 1st Number: 6
Enter the 2st Number: 5
The result of Addition is 11.0
The result of Subtraction is 1.0
The result of Multiplication is 30.0
The result of Division is 1.2
```

Conclusion:

A calculator that uses functions makes it easy to do basic math and reuse the code for future tasks.