**Task 8: Implement Python generators and Decorators**

**Task 8.1 : Produce the squares of numbers up to a limit**

**Aim:**  
To write a Python program that implements a generator to produce the squares of numbers up to a given limit.

**Algorithm:**

1. Start the program.
2. Define a generator function using the def keyword.
3. Inside the function, use a loop from 1 to n.
4. Use the yield statement to return the square of each number one by one.
5. In the main program, accept a number n from the user.
6. Call the generator function and iterate through it using a for loop.
7. Print the squares generated.
8. End the program.

Program

def square\_generator(n):

for i in range(1, n+1):

yield i \* i # yield returns values one by one

# Using the generator

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

print(f"Squares from 1 to {n} are:")

for val in square\_generator(n):

print(val)

Sample I/O

Enter a number: 5

Squares from 1 to 5 are:

1

4

9

16

25

**Task 8.2 : To show execution time of a function**

**Aim:**  
To write a Python program that implements a decorator to calculate and display the execution time of a function.

**Algorithm:**

1. Start the program.
2. Import the time module.
3. Define a decorator function that accepts another function as an argument.
4. Inside the decorator, define a wrapper function:
   * Record the start time.
   * Call the original function.
   * Record the end time.
   * Print the execution time.
5. Return the wrapper function from the decorator.
6. Use the @decorator\_name syntax to apply the decorator to a function.
7. Define a function (e.g., display\_numbers) that prints numbers with a small delay.
8. Call the decorated function.
9. End the program.

Program

mport time

# Decorator function

def timer\_decorator(func):

def wrapper():

start = time.time()

func()

end = time.time()

print(f"Execution Time: {end - start:.5f} seconds")

return wrapper

# Function to be decorated

@timer\_decorator

def display\_numbers():

for i in range(1, 6):

print(i)

time.sleep(0.5) # just to simulate delay

# Calling the decorated function

display\_numbers()

Sample I/O

1

2

3

4

5

Execution Time: 2.50012 seconds

**Task 8.3 Fibonacci sequence**

**Aim**

To design a Python program that implements a generator function fibonacci(n) which yields the first *n* Fibonacci numbers.

**Algorithm**

1. Start
2. Define a generator function fibonacci(n) that takes an integer n as input.
3. Initialize two variables:
   * a = 0 (first Fibonacci number)
   * b = 1 (second Fibonacci number)
4. Repeat the following steps for n iterations:
   * Yield the current value of a.
   * Update values:
     + a = b
     + b = a + b (sum of previous two numbers).
5. Outside the function, call the generator using for num in fibonacci(n):
6. Print each Fibonacci number as it is generated.
7. End

**Program:**

def fibonacci(n):

a, b = 0, 1

for \_ in range(n):

yield a

a, b = b, a + b

for num in fibonacci(10):

print(num, end=" ")

**Output**

0 1 1 2 3 5 8 13 21 34