Date: 17/9/25

Task 8: Implement python generator and Decorator

a) fibonacci sequence generator

Aim: to create a generator function that yields fibonacci numbers up to a given limit and display the sequence.

Algorithm:

1. Define a generator function fibonacci_generator (n) that taked a maximum value.
2. Initialize the first two fibonacci numbers (0 to 1).
3. Yield the first number (0)
4. use a while loop to generate subsequent fibonacci numbers.
5. Yield each fibonacci number until it exceeded the limit.
6. Get user input for the maximum value.
7. use the generate to iterate through and display the sequence.

Program:

```
def fibonacci_generator(n):
    """ generator function that yield fibonacci
        numbers upto n""""
    a,b = 0,1
    yield a
    while b <= n.
```

output:

enter the maximum value for fibonacci sequence : 50.

fibonacci sequence upto 50

0 1 1 2 3 5 8 13 21 34

```
        yield b
        a, b = b, a+b
    def main():
        try:
            n = int(input("enter the maximum
value for fibonacci sequence"))
        if n < 0:
            print("please enter a non-negative
                number").
            return
        print(f"Fibonacci sequence upto {n}:")
        fib_gen = fibonacci_generator(n)
        for num in fib_gen:
            print(num, end = " ")
        print()
        except value error:
            print("please enter a valid
            integer")
if __name__ == "__main__":main()
```

Result: Thus, the program successfully creates a generates function that produces fibonacci number upto specified limit.

5) function execution time Decorator

Aim: To implement a decorator that calculated and displays the execution time of any functions, specifically applied to sorting function.

Algorithm:

1. create a decorator function - timer - decoration that:
   - Records start time using time.time()
   - Calls the original function.
   - Records end time and calculates execution time.
   - Prints the execution time.
   - Returns the function result.
2. create a function sort - random - list(size) that:
   - Generates a list of random numbers.
   - Sorts the list using built-in sort
   - Returns the sorted list.
3. Apply the decorator to sorting function
4. Test the different list sizes.

Program:-

```
import time
import random
def timer - decorator (func):
    def wrapper (*args, **kwargs):
        start - time = time.time()
        result = func(*args, **kwargs)
        end time = time.time()
        execution time = end time - start - time.
```

output:-

Sorting list of size 1000:-

function 'sort_random_list' executed in 0.000998 second

list 5 elements: [2, 4, 6, 8, 10]

list 5 elements: [991, 992, 993, 995, 999]

Sorting list of size 5000:-

function 'sort_random_list' executed in 0.002995 seconds

list 5 element: [1, 1, 2, 2, 3]

list 5 element :- [998, 998, 999, 999, 1000]

```
print(f"Function '{func.__name__}' executed in
    {execution_time:
    return result
    return wrapper
@timer — decorator
def sort_random_list(size):
    random_list = [random.randint(1, 1000) for _ in
    range(size)]
    sorted_list = sorted(random_list)
    return sorted_list.
def main():
    size = [1000, 5000, 10000]
for size in sizes:-
    print(f"In sorting list of size{size}:")
    sorted_list = sort_random_list(size)
    print(f" first 5 elements: {sorted_list[:5]}")
    print(f" last 5 elements: {sorted_list[-5:]}")
if __name__ == "__main__":
    main()
```

VELTECH	
EX No.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	25
SIGN WITH DATE	

Result: Thus the decorator successfully measures and displays the execution time of the sorting functions are verified.