7/9/25

Task8: Implement Python generator and decorators.

(a). fibonacci sequence generator:

Aim:- To create a generator function that yields Fibonacci numbers up to a given limit n and display The sequence.

Algorithm:-

1. Define a generator function fibonacci-generator(n) That takes a maximum value.
2. Initialize the first two fibonacci numbers (0 and 1)
3. Yield the first number (0).
4. Use a while loop to generate subsequent fibonacci Numbers.
5. Yield each fibonacci number until it exceeds the limit n.
6. Get user input for the maximum value.
7. Use the generator to iterate through and display The sequence.

Program:-

```
def fibonacci_generator(n):
"""Generator function that yields fibonacci numbers
    up to n" """.
    a,b = 0,1
    yield a.
    while b<=n:
        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
```

output:-

```
Enter the maximum value for fibonacci sequence =5
Fibonacci sequence up to 50:
0 1 1 2 3 5 8 13 21 34.
```

```
            Print(f"fibonacci sequence up to {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 Generator function that Produces fibonacci Numbers up to the specified limit.

6. function execution time decorator.

Aim:- To implement a decorator that calculates And displays the execution time of any function, specifically applied to sorting function.

Algorithm:-

1. Create a decorator function timer_decorator that:
   - Records . start time using time. time( )
   - calls the original function.
   - Records and time and calculates execution time
   - prints the execution time.
   - Returns the function result.
2. Create a function ~~result.~~ 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:-

```
Inport 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.
        Print(f "function '{func...name--}' executed
        In {execution _time:-
```

Output:-

```
Sorting list of size 1000:
function 'sort_random_list' executed in 0.000998 sec
First 5 elements: [2, 4, 6, 8, 10]
Last 5 elements: [991, 992, 993, 995, 999]
Sorting list of size 5000:
Function 'sort_random_list' executed in 0.002995 se
First 5 elements: (1, 1, 2, 2, 3)
Last 5 elements: [998, 998, 999, 999, 1000].
```

```
        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 ():
    sizes =[1000, 5000, 10000]
    for size in sizes:
        Printf (f" \n 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 ()
```

VEL TECH - CSE	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	15
SIGN WITH DATE	[illegible] 15/10/25

Result:-

Thus, the decorator successfully measures and displays the execution time of the sorting function are, verified.

[illegible] 15/10/25