17/9/25

Task 8. 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.

Python program:-

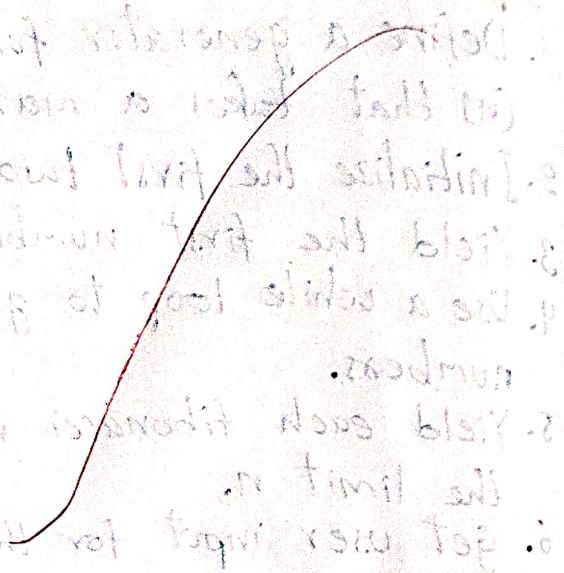
```
def fibonacci-generator (n):
""" Generator function that yields fibonacci numbers
    opto n """
    a,b = 0,1
    yield a.

    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: 50

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 ValueError:
        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 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.

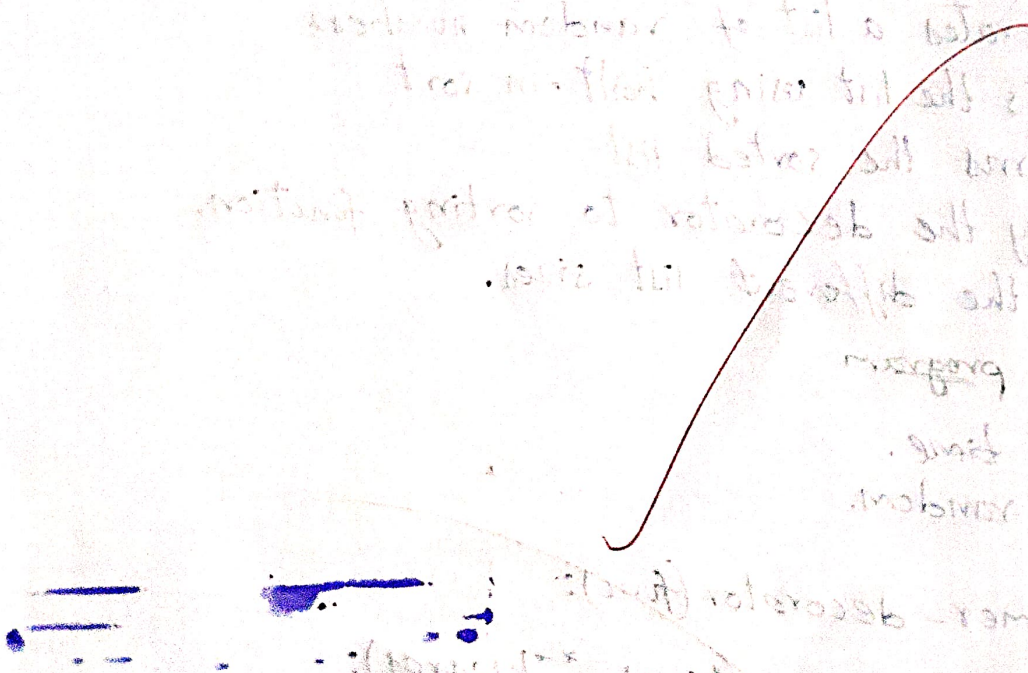
Python program

```
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.
        print(f"function '{func.__name__}' executed
          in {execution_time:.
        return result
    return wrapper.

@timer_decorator
def sort_random_list(size):
```

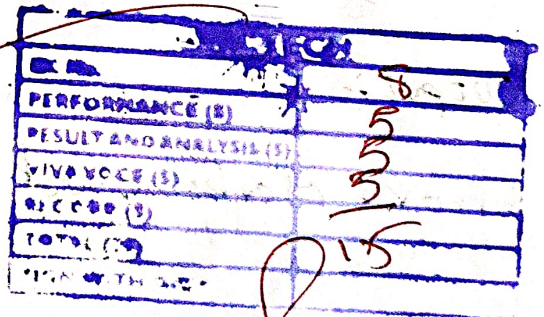
Output

Sorting list of size 1000:
function 'sort_random_list' executed in 0.000998 seconds
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 seconds
first 5 elements: [1, 1, 2, 2, 3]
last 5 elements: [998, 998, 999, 999, 1000].



```
    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:
        print(f"\nSorting 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()
```



14/10/25

Results:-

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

14/10/25