

8.	17/09/25	Implement Python Generators And Decorators	15	Aug 30/9/25
----	----------	---	----	-------------

# 7/9/25 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  $n$  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 upto 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  
        print(f"fibonacci sequence upto {n}:")  
        fib_gen = fibonacci_generator(n)  
        for num in fib_gen:  
            print(num, end = " ")  
        print()
```

Output:-

Enter the maximum value for fibonacci sequence = 50

Fibonacci sequence upto 50:-

0 1 1 2 3 5 8 13 21 34

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 upto the specified limit.



## 17/7/25 (b) 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 to get its execution time.
  - 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
        print(f"function '{func.__name__}' executed in {execution_time} s.")
        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]
```

Output:-

Sorting list of size 1000:

Sorting list of size 10000  
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]

[illegible]

$\text{shift} - \text{float} - \text{shift} - \text{int} = \text{shift} - \text{int} - \text{float}$   
 $\{ \dots \text{shift} - \dots \text{shift} \}$  without "if" being  
 $\dots \text{shift} - \text{float} - \text{int} \dots$   
 this is required

```

for size in sizes:
    print (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)	25
IN WITH DATE	

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