

17/11/20

## Task: 8 Implement python generator and Decorators

### a) Fibonacci sequence generator

Aim:

To write a python program using a generator function to yield Fibonacci numbers up to  $n$  and display the sequence

Algorithm:

1. Start the program
2. Define a generator function `fibonacci(n)` that:
  - initialize first two numbers  $a=0, b=1$ .
  - uses `yield` to generate Fibonacci numbers
  - update values in each iteration
3. Accept input  $n$  from the user.
4. call the generator function and display numbers
5. stop the program

Program:

```
def fibonacci(n):
```

```
    a, b = 0, 1
```

```
    for i in range(n):
```

```
        yield a
```

```
        a, b = b, a + b
```

```
n = int(input("Enter how many Fibonacci numbers to generate: "))
```

```
print("Fibonacci sequence:")
```

```
for num in fibonacci(n):
```

```
    print(num, end=" ")
```

Result:

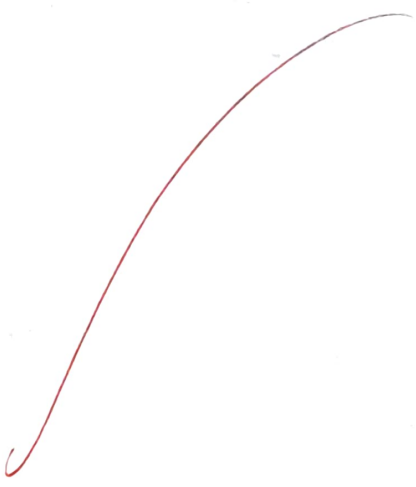
Thus, the program for generating Fibonacci sequence using a generator function was successfully created.

output:

Enter how many Fibonacci  
number to generate : 7

Fibonacci sequence :

0 1 1 2 3 5 8



## b. Function Execution time decorator

Aim:

To write python program using a decorator that calculates the execution time of any function and apply it to a function that sorts a list of random numbers.

Algorithm:

1. Start the program
2. Import time and random module
3. Define a decorator function execution\_time(func) that:
  - Records start time before function call
  - Executes the function.
  - Records end time and prints execution time
4. Define a function sort\_numbers() that generates random number and sorts them.
5. Apply the decorator to the function
6. Call the decorated function
7. Stop the program

Program:

```
import time
import random
```

```
def execution_time(func):
```

```
    def wrapper(*args, **kwargs):
```

```
        start = time.time()
```

```
        result = func(*args, **kwargs)
```

```
        end = time.time()
```

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

```
        return result
```

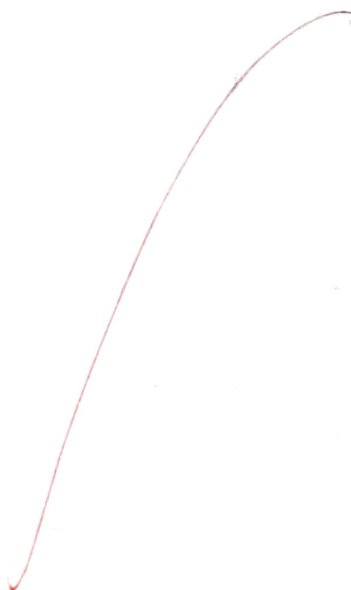
```
    return wrapper
```

3.

Sample output:

Sorting completed

Execution Time : 0.002131 seconds



@ execution - time

```
def sort_numbers():
```

```
    numbers = [random.randint(1,1000) for _ in  
                range(10000)]
```

```
    numbers.sort()
```

```
    print("Sorting completed!")
```

```
sort_numbers()
```

EX NO	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	15

Result:

Thus the program using a decorator to calculate execution time of a function was successfully executed and applied to a sorting function.