10-9-25.

Implement python generator and decorators

Task 7

Aim: write a python program to implement python generator and decorators

Algorithm

1. Define generator function:
   * Define the function number - sequence

2. Initialize current value:
   * set current to value of start

3. generate sequence:
   * yield the current value of current
   * Increment current by step

4. Get user input
   * Read the starting no from user input
   * Read the ending no from user input
   * Read the step value

5. Create generator object:
   * create a generator object number - sequence with user - provided values

6. Print generated sequence:
   * Print each value

```
def number - sequence (start, end, step = 1):
    current = start
    while current < = end:
        yield current
        current + = step
start = int (input ("Enter the starting number: "))
```

Output:

```
Enter the starting no : 1
enter the ending no : 50
enter the step value : 5
1
6
11
16
21
26
31
36
41
46
```

```
end = int(input("Enter the ending number:"))
step = int(input("Enter the step value:"))
# create the generator
sequence_generator = number_sequence(start, end, step)
# print the generated sequence of numbers
for number in sequence_generator:
    print(number)
```

Produce a Default sequence of no starting from 0, ending at 10, with step of 1

Algorithm

1. start function
   * Define the function my-generator (n)
2. initialize counter:
   * set value at 0.
3. generate values
   * yield the current value
   * increment value by 1
4. create generator object:
   * call my generator(11) to create generator
5. Iterate and print values:
   * Print value

8.1 (b) Program

```
dee my-generator (n):
    # initialize counter
    value =0
# loop until counter is less than n
    while value < n:
# produce the current value of the counter
    yield value
# increment the counter
    value += 1
# iterate over the generator object produced by
    my-generator
for value in my-generator (3):
# print each value produced by generator
    ~~out put~~ print (value)
```

12 imagine you are working on a messaging application that needs to format message differently. Based on the user's preference users can choose to have their message

Algorithm

1. create Decorators:
   * Define upper case - decorator to convert the result
   * Define lower case - decorator to convert the result lower case
3) Define function
4) execute the program

Program

```
def uppercase. - decorator (func):
    def wrapper (text):
        return func (text). upper ()
    return wrapper
Define lower case - decorator (func):
    def wrapper (text):
        return. wrapper
@ upper case - decorator
    def shout (text):
        return text
@ lower case - decorator
    def whisper (text):
        return text
```

```
def greet(func):
greeting = func("Hi, I am function passed
                as an argument.")

greet(shout)
greet(whisper)
```

VEL TECH	
EX NO.	7
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	15
SIGN WITH DATE	

Result: Thus, the Python program to implement Python generator and Decorators was successfully executed and the output verified