Task-7: Implement Python generator and decorators.

Aim: write a Python Program to implement Python generator and decorator.

Q.1 write a Python Program that includes a generator function to Produce a sequence of numbers. The generator should be able to:
a) Produce a sequence of numbers when Provided with start, end, and step values.
b) Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are Provided.

Algorithm:

1. Define Generator Function
→ Define the function number_sequence (start, end, step=1)
2. Initialize current value:
→ set current to value of start.
3. Generate sequence:
→ while current is less than or equal to end.
→ yield the current value of current
→ Increment current by step.
4. Get user Input:
→ Read the starting number (start) from user input.
→ Read the ending number (end) from user input
→ Read the step value (step) from user input.
5. Create Generator object:
→ Create a generator object by calling number_sequence (start, end, step)
6. Print Generated sequence:
→ Iterate over values Produced by generator object.
→ Print each value.

Program:

```
def number_sequence (start, end, step=1)
current = start
while current <= end:
 yield current
 current += step
start = int (input ("enter the starting number:"))
end = int (input ("enter the ending number:"))
step = int (input ("enter the step value:"))
```

```
# Create the generator
sequence_generator = number_sequence
# Print the generated sequence of numbers.
for number in sequence_generator:
    print(number)
```

Output:

```
Enter the starting number: 1
Enter the ending number: 50
Enter the step value : 5.
1
6
11
16
21
26
31
36
41
46
```

7.1 (b): Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided.

Algorithm:

1. Start Function:
→ Define the function my_generator(n) that takes a Parameter n.
2. Initialize counter:
→ set value to 0.
3. Generate values:
→ while value is less than n:
    * Yield the current value.
    * Increment value by 1.
4. Create Generator object:
→ Call my_generator(11) to create a generator object.
5. Iterate and Print values:
→ For each produced by generator object.
→ Print value.

Program:

```
def 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.
        Print (value).
```

Output:

```
0
1
2
```

Q.2 Imagine you are working on a messaging application that needs to format messages differently based on user's preferences. User can choose to have their messages automatically converted to upper case or to lower case. You are provided with two decorators: upper case_decorator and lower case_decorator. These decorators modify the behavior of functions they decorate by converting the text to upper case or lower case, respectively. write a program to implement it.

Algorithm:

1. Create Decorators:
   → Define uppercase_decorator to convert the result of a function to uppercase.
   → Define lower case_decorator to convert the result of a function to lower case.
2. Define functions:
   → Define shout function to return input text. Apply the function
   → Define whisper function to return the input text.
3. Define Greet Function:
   → Define greet function that:
      * Accepts a function as input.

* Prints the result.

4. Execute the Program:
→ Call greet to print the greeting in uppercase.
→ Call greet to print the greeting in lower case.

Program:

```
def uppercase_decorator(func):
def wrapper(text):
return func(text).upper()
return wrapper
def lowercase_decorator(func):
def wrapper(text):
return func(text).lower()
return wrapper
@uppercase_decorator
def shout(text):
return text
@lowercase_decorator
def whisper(text):
return text
def greet(func):
greeting = func("Hi, I am created by a function passed as an argument.")
greet(shout)
greet(whisper)
```

output:

HI, IAM CREATED BY A FUNCTION PASSED AS AN ARGOMENT.
hi, iam created by a function passed as an argument.

...ORMANCE (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 was verified.