Task 8:- implement Python generator and decorators

Aim:- write a Python program to implement Python generator and decorators

8.1 write a Python program that includes a generator function to produce a sequence of numbers the program 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 if no value are provided

Produce a sequence of numbers when provided with strat, end, and step values.

Algorithm:-

1) Define Generator function
   - Define the function number_sequence (start, end, step=1)
2) initialize current value
   - set current to the 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.

out put:-

```
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
```

5) create Generator object
- create a generator object by calling number_sequence (start, end, step) with user-provided values

6) print Generated sequence
- iterate over the values produced by the generator object
- print each value.

8.1 program

```
def number_sequence(start, end, step=1):
    current = start
    while current <= end
    Yield 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(start,end,
    step).
    # print the generator sequence of numbers
    for number in sequence_generator:
        print(number)
```

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-gen generator[n] that takes a parameter n.

2) initialize counter
- set value to 0

output :- 0
1
2

3) Generate values
- while value is less than n
  - yeld the curred value
  - increment value by 1

4) Create generator object
- call my-generator [17] to create a generator object

5) Iterate and print values
- for each values produced by the generator object
  - print value

8.1 (b) Program

```
def my-generator [n]:
# intialize 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
# # 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]
```

8.2 Imagine you are working on a messaging application that needs to format massages differently based on the user's perforences users can choose to have their messages automatically converted to uppercase [for emphasis] or to lower-case [for a softer tone]. you care provided with two decorates uppercase-decorator and lowercase-decorator. these decorators modify the behaviour of the functions they decorate by converting text to upper case for lowercase respecting. write a program to implement it.

Algorithm:-

1) create decorators:
   - Define uppercase-decorator to convert ~~of~~ the result of a function to uppercase
   - Define lowercase-decorator to convert the result of a function to lower case.

2) Define functions:
   - Define shout function to return the input text Apply @ uppercase-decorator to this function.
   - Define whispey function to return the input text apply @ lowercase-decorator to this function.

3) Define Greet function:
   - Define greet function that:
     - Accepts a function [func] as input
     - calls this function with the text "Hi, I am created by a function passed as an argument"
     - Print the result

output- HI, IAM CREATED BY FUNCTION PASSED AS AN ARGUMENT,
hi, i am created by a function passed as an argument

- call greet [shout] to print the greeting in uppercase
- call greet [whisper] to print the greeting in lowercase.

Program:-

```
def uppercase_decorate [func]:
  def wrapper [text]:
    return func[text].upper[]
  return wrapper
  def lowercase_decorate [func]:
    def wrapper [text]
      return func [text].lower[]
    return wrapper
@ uppercase_decorator
  def shout [text]:
      return func(text).lower[]
  return wrapper
      return text
@ lowercase_decorater
  def whisper [text]:
      return text

  def greet [func]:
    greeting = func("Hi, I am created, by a
    function passed as an argument")
    print [greeting]
    greet [shout]
    greet [whisper]
```

VEL TECH	
EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RECORD (5)	
TOTAL (30)	
SIGNATURE	

Result:- Thus, the Python Program to implement Python generator and decorators was successfully executed and the output was verified.