Task8: 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.

a) produce a sequence of numbers when provided with start, end and step values.

b) produce a default sequence of number starting from 0 ending at 10 and with a step of 1 if no value are provided.

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.
5. Create Generator object:-
   * create a generator object by calling number_sequence( start,end,step) with
     user_provided values.
6. Print Generated sequence:
   * Interate 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:
         yeild current
         current+=step
 start=int(input("Enter the starting number:"))
 end=int(input("Enter the ending number:"))
 step=int(input("enter the step valued:"))
 sequence_generator=number_sequence(start,end,step)
   for number in sequence_generator:
       print(number)
```

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:
```

output:

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

o/p:

```
    *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 value produced by the generator object:
    * print value.
```

8.1(b) program:

```
 def my_generator(n):
     value = 0
     while value < n:
       yield value
       value+= 1

   for value in my_generator(3):
   print (value).
```

8.2 Imagine you are working on a messaging application that needs to format message differently based on the user's preferences.

Algorithm:

```
1. Create Decorators:-
   *Define uppercase_decorator to convert the result of a function to uppercase.
   *Define lowercase_decorator to convert the result of a function to lowercase.
2. Define functions:-
   * Define shout function to return the input text. Apply @ uppercase_decorator to
     this function.
   *Define whisper function to return 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.
4. Execute the program:-
   * call greet (shout) to print the greeting in upper case.
   *call greet (whisper) 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):
```

Output:

```
0
1
2
```

o/p

output:

```
HI, I AM CREATED BY A FUNCTION AS AN ARGUMENT.
hi, iam created by a function passed as an argument.
```

[illegible]

```
return func(text).lower()
return wrapper
@uppercase_decorator
 def shout(text):
  return wrapper
 @lower case_decorator
  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.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	
SIGN WITH DATE	[illegible]

Result: Thus the python program to implement Python generator and decorators was successfully executed and the output was verified.