Task 8: Implement python generator and decorators

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

Q) 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 numbers starting from 0, ending at 10, and with a step of 1 if no values are provided

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

Algorithm:

1. Define Generator Function:
   - Define the function ~~print~~ number_sequence (start, end, step=1)

2. Initialize current value:
   - set current to the value the start

3. Generate Sequence
   - while current is less than or equal to end:
     - yield the current value of current
     - increment current by step.

4. Create Generator object:

- Create a generator object by calling number-sequence with user-provided values

5. Print Generated Sequence:

- Iterate over the values provided by the generator object
- print each value

<u>8.1 program</u>

```
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 generator 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
```

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 value produced by the generator object:
        - Print value.

8.2 (b) 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
```

Imagine you are working on a messaging application that needs to format message differently based on the user's preferences. Users can choose to have their messages automatically converted to uppercase ~~or~~ to uppercase - decorator and lower case - decorator. write a program to implement it.

Algorithm:

1. Create Decorators:
   - Define uppercase - decorator to convert the result of a function to upper case
   - Define lower case - decorator to convert the result of a function to lower case.
2. Define Functions:
   - Define shout function to return the input text. Apply @upper case - decorator to this function
   - Define whisper function to return the input text. Apply @lowercase - decorator to this function.
3. Define greet Function:
   - Define greet function that:
     - Accepts a functions as input.
     - calls this function with the text "Hi. I am created by a function passed as an argument."
     - prints the result.
4. Execute the program
   - call greet to print the greeting in upper case
   - call greet to print the greeting in lower case.

Program:-

```
def upper case -decorator (func):
    def wrapper (text):
        return func(text). upper()
    return wrapper

def lower case -decorator (func):
    def wrapper (text):
        return func(text)- lower()
    return wrapper

@ uppercase -decorator
def shout(text):
    return text

@ 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)	[illegible]
SIGN WITH DATE	[illegible] 17/9

Result:- Thus the python programs Implemen
python generator and decorators was
successfully executed and the output
was verified.

output:-

```
HI . I AM CREATED BY A FUNCTION
PASSED AS AN ARGUMENT

hi, i am created by a function passed
as an argument
```