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 generator should be also to.

a) Produce a sequence of numbers when provided with start, 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 that 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.

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
```

6. Print Generator Sequence

- Iterate over the values produced by the generator object.
- Pitch 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(intput("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)
```

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

Algorthm

```
1. Start Function:
    o Define the function my-generator (n) that takes
      a parameter n.
2. Initialize Counter.
    o set value to 0.
3. Generate values.
    o While value is less than n.
        Yield the current value
        Increment value by 1.
```

Output:

```
0
1
2
```

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.

Program:

```
def my_generater(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_generater(3).
   #print each value produced by generator.
   print(value)
```

8.2: Imagine you are working on a messaging application that needs to format messages differently based on the user's performce. User can choose the have their messages automatically converted to upper case (for emphasis) or ~~lower~~case (for a softer tone). You are provided with two ~~dive~~ decorators: uppercase_decorater and lowercase_decorator. These decorators modify the behavior of the functions they decorate by converting the text to uppercase or lowercase, respectively. write a program to implement it.

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 function.
    - Define shout function to return to convert the input text. Apply @uppercase_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 function (func) 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 (shout) to print the greeting in uppercase
    - Call greet (whispker) to print the greeeting in lowercase.

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 (func)~~
~~def shout (text):~~
~~    return text~~

~~@ lowercase_decorator~~
~~def whisper (text):~~
~~    return text~~

~~def greet (func):~~
~~    greeting = func ("Hi, Iam created by a function~~
~~    passed as an argument~~
```

Output:

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

```
@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")
    print(greeting

greet(shout)
greet(whisper)
```

VELTECH	
EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (3)	
VIVA VOCE (3)	
RECORD (4)	
TOTAL (15)	
SIGN WITH DATE	

Result:

Thus the python program to implement python generator and decorators was successfully excuted and the output was verifed.