Task-7 Implementation python generator and decorators

Aim: Write a python program to implement python generator and decorators

7.1. 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 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 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:

- Iterated over the values produced by the generator object
- Print each value.

7.1 Program:-

```
def number_sequence(start, end, step=1):
    current = start
    while current <= end:
        yield current
        current += step
step = int(input("Enter the starting number :"))
end = int(input("Enter the Ender number:"))
set = int(input("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)
```

Output:

Enter the starting number : 1

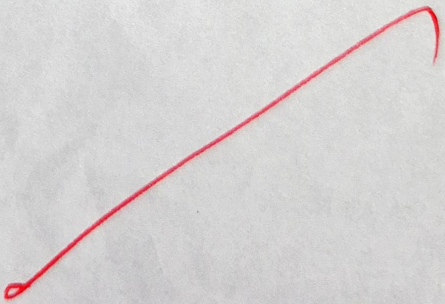
Enter the ending number : 50

Enter the step value : 5

1
6
11
16
21
26
31
36
41
46

Output:

0
1
2



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_generation(n) that takes a parameter n.
2. Initialize counter:
   - Set value to 0
3. Generate values:
   - while values 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.

7.1(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 computer.
 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)
```

8.2 Imagine you are working on a messaging application that needs to formate messages differently based on the user's preferences. User can choose to have their messages automatically converted to uppercase (for emphasis) or to lowercase (for a softer tone). You are Provided with two decorators: uppercase_decorator and lowercase_decorator. These decorators modify the behaviour of the functions they decorate by converting the text to uppercase or lowercase, respectively. Write a program to implement it.

Algorithm:

1. Create Decorators:
   - Upper Case_decorator to convert the result of a function to uppercase.
   - Define ~~lower_case~~ lowercase_decorator to convert the result of a function to lowercase.

2. Define Function:
   - Define shout function to return the input text. Apply ~~@upper_case~~ ~~@Upper_case~~ @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 greetings in uppercase.
   - call greet(whisper) to Print the greeting in lowercase.

output:

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT

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

Program:

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

VEL TECH - CSE	
EX NO.	1
PERFORMANCE (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