Task-8: implement python generators and decorators

Aim: write a python program to implement python generators 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 start, end, and step values

Algorithm:

1. Define Generator function
   - define the function number. sequence (start, end, step=1)
2. initialize current values
   - set current to the values 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

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 stopping value:"))
    step = int(input("enter the step value:"))
    # 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:

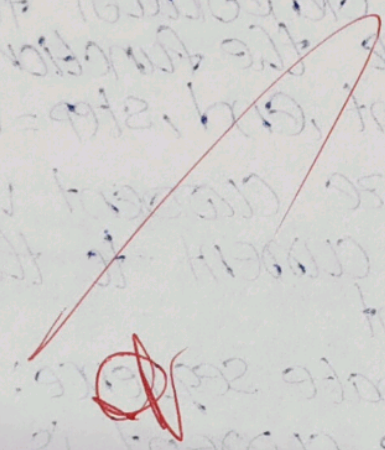
1) start function:
- Define the function my_gen generator[n] that takes a parameter.

2) initialize counter
- set value too

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



3) Generate values
- while value is less than n
  - yield the cubed 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]:
# 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[5]
# print each value in my_generator
        print [value]
```

output: 0
1
2

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

Algorithm:

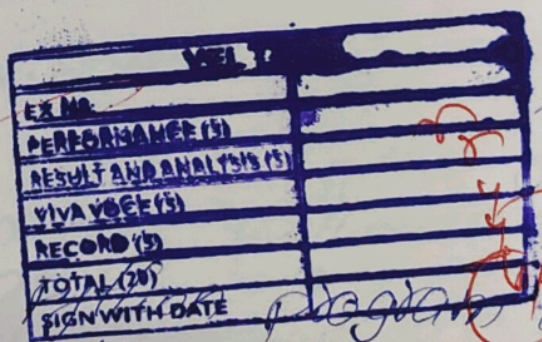
1) Create decorators:
   - Define uppercase_decorator to convert the result of a function to uppercase.
   - Define lower case_decorator to convert the result of a function to lower case.
2) Define functions:
   - Define shout function to return the input text Apply @ uppercase_decorator to this function
   - 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
     - print the result

output: HI, I AM CREATED BY FUNCTION PASSED
AS AN ARGUMENT
hi, i am created by an function passed
as an argument

4) execute the program:
- 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 wrapped [text]
     return func [text].lower[ ]
    return wrapped
@ upper case_decorated
 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]
```



result: Thus the program to implement python generator & decorator was successfully executed and the output was verified.