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 number.

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

b. 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. 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:
     - yeild the current value of current.
     - Increment current by step.
4. Get user input:
   - Read the starting number from user input
   - Read the ending number from user input
   - Read the step value from user input
5. 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
```

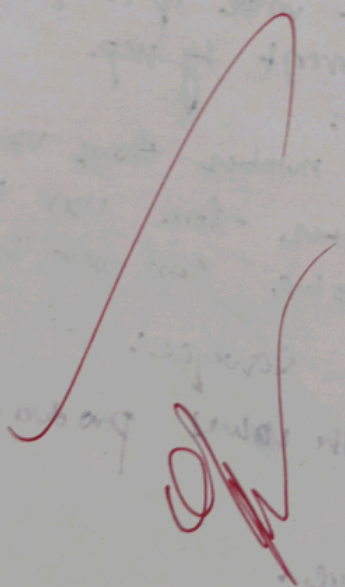
output

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



```
    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 number:"))
sequence_generator = number_sequence(start, end, step)
for number in sequence_generator
    print(number)
```

b) Algorithm:-

1. Start
   - 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.
     - yeild the current value.
     - Increment value by 1.
4. Create Generator object:
   - Call my_generator(n) 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 messages differently based on the user's preferences. user can choose to have their messages automatically converted to uppercase or to lowercase. you are provided with two decorators:

Algorithm:-

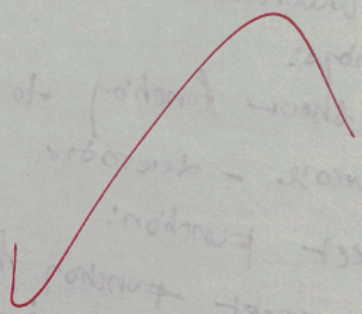
1. Create Decorators:
   - Define uppercase - decorator to convert the result of a function to uppercase.
   - Define lowercase - decorator to convert the result of a function to lowercase.
2. Define functions:
   - Define shout function to return the input text.
   - Apply @uppercase - decorator to this function.
3. Define Greet function:
   - Define greet function that
     - Accepts a function 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 (whisper) to print the greeting in lowercase.

Program:-

```
def uppercase_decorator(func):
    def wrapper(text):
        return func(text).upper()
    return wrapper
def lowercase_decorator(func):
    def wrapper(text):
```

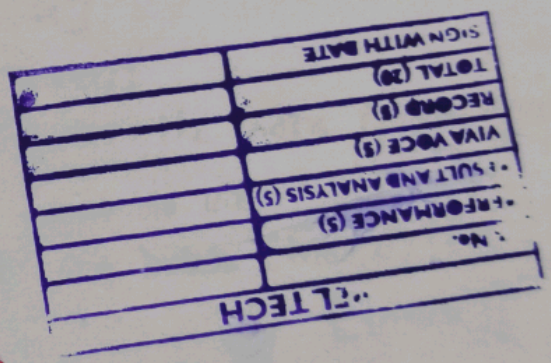
Output:- HI, I AM CREATED BY A FUNCTI
PASSED AS AN ARGUMENT

Hi, i am created by a function passed as an argument.



```
        return fun(text).lower()
    return wrapper
@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)
```

VEL TECH	
EX No.	5
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	4
RECORD (5)	4
OTAL (20)	
.N WITH DATE	



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