TASK 8: Implement python generator and decorators.

Aim:-

Write a python program to implement python generator and decorators.

Algorithm:-

1. Define GeneratorFunction:
   - define the Enction 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 from user input
   - Read the ending number from user input
   - Read the step value from user input.
5. Create Generator object.
   - Create a generator object by calling number_sequence with user provided values.
6. Print Generated sequence:
   - Iterate over the values prodidced by the generator object.
   - print 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(input("Enter the step value:"))
```

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
```

```
#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 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.

(b) Program :-

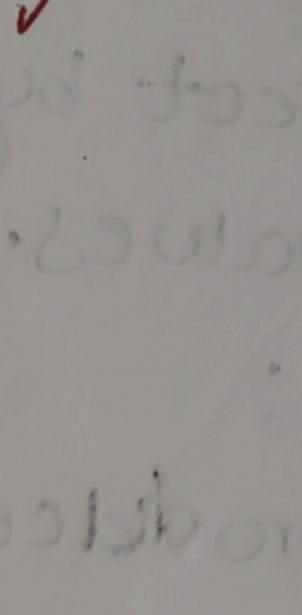
```
def my_generator(n):
    #initialize counter
    value = 0
    # loop until counter is less than n
    while value < n:
        #Produce the current value of the counter
        yield value
        #increment the counter
        value += 1
#iterate over the generator object produced by my_generator
for value in my_generator(3):
    # Print each value produced by generator
    print(value)
```

output:-

0

1

2



8.2:-

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.
   - 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(whisper) to print the greeting in lowercase.

Program:-

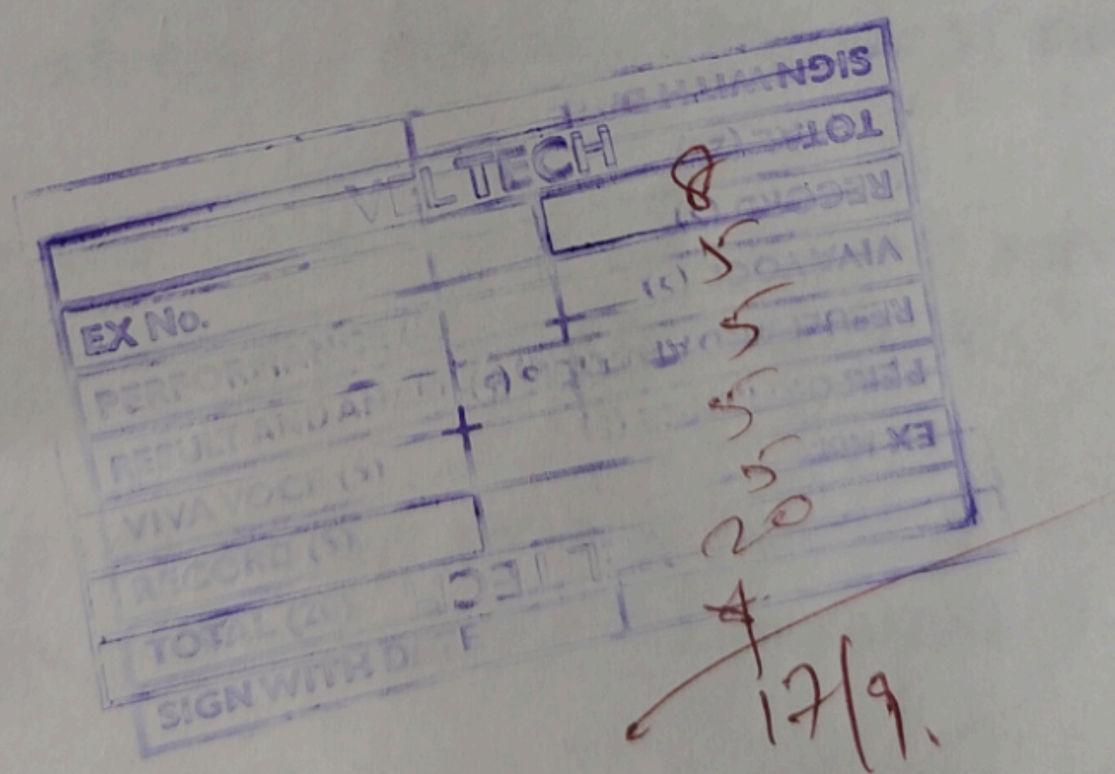
```
def uppercase_decorator(func):
  def wrapper(text):
    return func(text).upper()
  return wrapper
def lowercase_decorator(func):
  def wrapper(text):
    return func(text).lower()
  return wrapper
@uppercase_decorator
def shout(text):
  return text
@lowercase_decorator
def whisper(text):
  return text
```

Q2:-

Output:-

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.
hi, iam created by a function Passed as an argument:

```
def greet(func):
    greeting = func("Hi, I am created by a function pass
    -ed as an argument.")
    print(greeting)
greet(shout)
greet(whisper)
```



Result:-

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