Task:-8 Date :- 15/9/25

Task 8 :- Implement python generator and decorator

Aim:- Write a python program to implement python genarator and decorators.

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

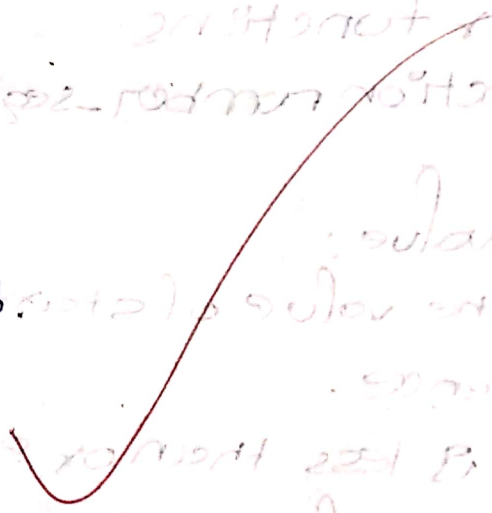
Algorithm:-

1. Define Generator functions:
   - 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 or 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 objects:
   - Create a genera tor abject by calling number_sequence (start, end, step) with user-Provided values.
6. Print Generated sequences.
   - iterate over the values produced by the generator object.
   - print each 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.



Program:-

```
def num - 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 i")).
## create the generator.
  sequence - generator = number_sequence(start,end,step
# Print the generated sequence of numbers for
number in sequence_generator:
  print (number).
```

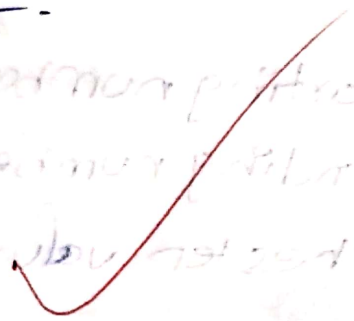
b) Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided.

Algorithm:

1. Start function:
   - Define the function my- generator(n) that takes a parameter n.
2. initialize counter:
   - Set value to 0.
3. Create 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.

output:-

0
1
2.



5. iterate and print values.

, for each value produced by the generator.

object:

, print value:

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.
# Print each value produced by generator
    Print (value)
```

8.2. imagine you are working. there 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:
   - 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

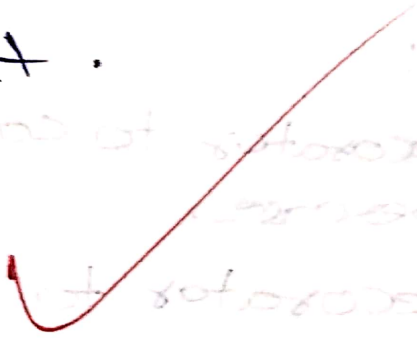
Program

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

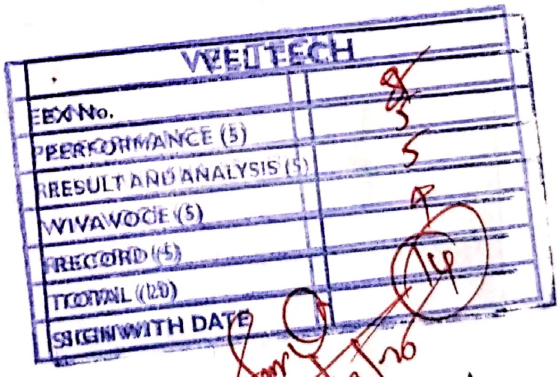
output:-

HI , IAM CREATED BY A FUNCTION PASSED AS AN ARGUMENT

hi , iam created by a function passed as an argument .



```
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.
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 python program to implement Python generator and decorators was successfully executed and the output was verified

Output :-