Task 8 :- Implement python
- toos

Aim :- Write a Python program to implement Python generator and decoratoos

8.1 :- 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 of 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 is ~~less than~~ (or) 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 point.

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
```

- 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:
   - 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
        current += step
start = int (input ("Enter the starting number:"))
end = int (input ("Enter the ending number:"))
step = 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)
```

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

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.

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(3):
# Print each value Produced by generator Print
    (value)
```

OutPut :-

0

1

2

8.2:- Imagine You are working on a messaging application that needs to Format messages differen-tly based on the user's preferences. users can Choose to have their messages automatically con-verted to uppercase (or) to lower case. 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 short Function to return the input text. Apply @ uppercase_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."
     - Points the result.
4. Execute the Program:
   * Call greet (short) to Print the greeting in uppercase.
   * Call greet (whisper) to Print the greeting in lowercase.

Program:-

```
def uppercase_decorator(func):
```

output :- HI, I AM CREATED BY A FUNCTION PASSED AS AN ANGUMENT.

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

```
    return Func(text).upper()
  return wrapper
def lower case_ decorator (Func):
  def wrapper lower case (text):
    return Func(text).lower()
  return wrapper
@upper case_decorator
def shout (text):
    return text
@lower case_decorator
def whisper (text):
  return text
def greet (Func):
    greeting = Func ("Hi, I am created by a Funct-
                      -ion passed as an argument.")
    Print (greeting)
greet (shout)
greet (whisper)
```

VEL TECH	
Ex. No.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	5
TOTAL (20)	20
SIGN WITH DATE	

Result:- Thus, the program to implement Python gen-
-erator and decorators was successfully executed
and the output was verified.