Task 8:- Implement Python generator and decorator

Aim:-
write a Python Program to implement Python generator and decorators

8.1:- write a Python 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 starting 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 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 input.
- 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.

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
```

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) take a parameter n.
2. Initialize counter:
   - set value to 0
3. Generate values:
   - while values 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.

output:

0
1
2

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).
```

8.2 Imagine you are working on a messaging application that needs 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 (for a softer tone).

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 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 @ lowcase - decorator to this function.
3. Define Greet Function:-
   - Define greet function that:
   - Accepts a function with the text "Hi, I am" created by a function passed as an argument.
   - Prints the result.

output:

HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT

hi, i am created by a 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 lower case.

Program:

```
def uppercase-decorator (func):
    def wrapper (text):
        return fun ((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
def greet (func):
    greeting = func("Hi, I am created by a function
    passed as an argument.")
    print(greeting)
greet(shout)
greet(whisper)
```

VIVA
SI. No.
PERFORMANCE (5)
RESULT AND ANALYSIS (5)
VIVA VOCE (5)
RECORD (5)
TOTAL (20)
SIGN WITH DATE

Result:-

Thus the Python Program to implement Python generator and decorators was successfully executed and the output was verified.