15-07-25 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 numbers. The generator should be able to:

a) Produces 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) Intialize 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 the user input

5) Create Generator object:
- Create generator object by calling number_sequence (start, end, step) with user - Provided values.

6) Print Generated Sequence:
- Iterate over the value 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 takes 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:-

```
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(b) Program:

```
def my_generator (n):
  # intialize counter
    value = 0
# loop until counter is less than n
  while value <n:
# produce the current value of the counter
yield value
   # ~~Provide the current value~~ o.
   # 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 message differently based on the user's preferenc users can choose to have their message automatically converted to upper case (for emphasis) or to lower case (for a softer tone). you a Provided with two decorators: upper case_ decorator and lower case_de These decorators modify the behaviour of the functions they decorat by converting the text to upper case or lowercase, respectively. write Program to implement it.

Algorithm:

1) Create Decorators:

- Define upper case _decorator to convert the result of a function upper case.
- Define lower case _decorator to convert the result of a function lower case.

2) Define Functions:

- Define should function to return the input text. APPly @ upper case _decorator to this function.
- Define whisper function to return the input text. APPly @ lower case _decorator to this function.

output:

```
HI, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT
hi, i am created by a function passed as an argument.
```

3) Define Greet function:

- Define greet function that
- Accepts a function (func) as input.
- Call this function the text "Hi, I am created by a function passed as an argument!!
- Print the result.

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 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
@lower case_decorator
 def shout (text):
  return text
def greet (func):
 greeting =func ("Hi, I am created by a function passed as an arugemen
 Print (greeting)
 greet (shout)
 greet (whisper)
```

VEL TECH - CSE	
EX NO.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	
SIGN WITH DATE	15

[illegible]

Result:- Thus the Python Program to ~~find~~ implement Python generate and Decorators was successfully executed and the output was verified.