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.

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

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

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
```

O/P

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)
```

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.

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 message differently based on the user's preferences.

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 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".
   * print the results.
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
```

Output:-

0
1
2

O/P

Output:-

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

```
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 are successfully executed and the output was verified.

VEL TECH	
[illegible]	
[illegible] PERFORMANCE (5)	
[illegible] RESULT AND ANALYSIS (5)	
[illegible] VIVA VOCE (5)	
[illegible] RECORD (5)	
[illegible] TOTAL (25)	
[illegible] SIGN WITH DATE	