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. 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 value 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 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 sequences:

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

- Iterate over the values produced by the generator object.
- print each value.

8.1 Program:-

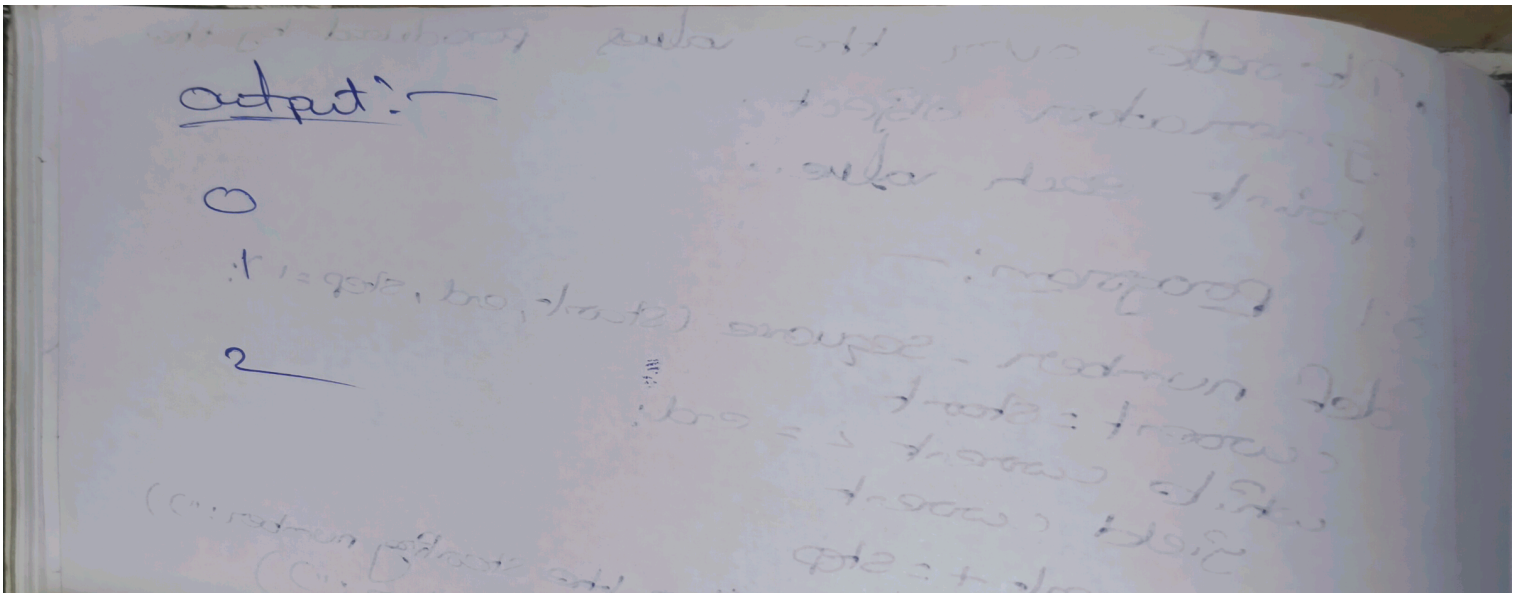
```
def number_sequence(start, end, step=1):
    current = start
    while current <= end:
        yield current
        current += step

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)
```

procedure 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 take a parameter n.
2. Initialize counter:
   * set value to i
3. Generate values.
   * while value is less than n.
     - yield the current value.
     - Increment value by 1
4. create generator object:
   * call my_generator(4) to create a generator object.
5. Iterate and print values:
   * for each values produced by the generator object.


output:-
0
1
2

• 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 messages differently based on the user's perference. users can choose to have their messages automatically converted to uppercase or to lowercase. you are provided 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 shout 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.';

- prints the result.

4. execute the program.

* call greet (shout) to print the greeting in upper case.

* call greet (whisper) to print the greeting in lower case.

program:-

```
def uppercase_decorator(func):
    def upper(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
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 program to implement Python operator and decorators was successfully executed and the output was verified.

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