Task-8 :- Implement python generator and decorators

Aim :- Write a python program to Implement python generators and decorators

8.1 :- Write a python program that includes a generator function to Produce a sequence of number.

a. Produce a sengence 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 non values are provided.

Algorithm :-

1. Define Generator Function:
   * Define the function number_sequence (start, end, step=1)
2. Initialize Cussent value:
   * set cussent to the value of start
3. Generate Sequence:
   * While cussent is less than or equal to end:
     -) yield the cussent value of cussent
     -) Increment cussent 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 users input.
5. Create Generator object:
   * Create a generator object by calling number_sequence (start, end, step) with users provided values
6. Print Generator Sequence:
   * Iterate over the values produced by the generator object
   * Print each value.

P.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 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. Generator values:
   * While value is less than n
     → yield the current value
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(4 Program :- (b)

```
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 counted
        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 upper case 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 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 function passed as an argument."
     -> Prints the result.

Output :-

0

1

2

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

hi . i am created by a function passed as an argument

8.2 :- 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
@ 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 function passed as an argument.")
    Print(greeting)
greet.(shout)
greet(whisper)
```

Result :- Thus, the program to Implement python generators and decorators was successfully executed and the output was verified.