Date: 17/9/25

Task 8 - Implement Python generator and decorators

Aim: Write a python program to implement python generator and decorator.

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 (end) from user input.

Read the step value (step) from user input.

5. Create Generator object

By calling number_sequence (start, end) with user-Provided values.

6. Print Generated sequence

Iterate over the values produced by the generator object.

Print each value.

Program:

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

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)

Aim: To write python program my_generators using loop statement.

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 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 values.

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
```

```
# Iterate over the generator object produced by my_generator
for value in my_generator(3):
    # Print each value produced by generator
    print(value)
```

output:

0
1
2

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 or to lowercase. You are provided with two decorators: uppercase decorator and lowercase_decorator. These decorators modify the behavior of the functions they decorator by converting the text to uppercase or lowercase, respectively. Write a Program to implement it.

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 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"
     - Prints the result.
4. Execute the program:
   * Call greet (shout) to print the greeting in uppercase
   * Call greet (whisper) to print the greeting in lowercase.

Output:

Hi, I AM CREATED BY A FUNCTION PASSED AS AN ARGUMENT.

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