Task 8.1(a) Implement python generators & decorators

17/9/25

Aim: To produce a sequence of numbers when provided with start, end & step values

Algorithm:

- Define the function number_sequence (start, end, step=1)

2. Initialize Current value
   - Set current to 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 string number (start) user input.
   - Read the ending number (end) user input
   - Read the value (step) from input
5. Crate Generator Object:
   - Crate generator obj by calling number_sequence (start, end, step)
6. Print Generated Sequence:
   - Iterate over the values produced by generator object
   - Print each value.

Program:

```
def number_sequence (start, end, step=1):
    current = start
    while current <= end
        Yield current
        current += step
start = int(input("Enter the staring number:"))
end = int(input("Enter the ending number:"))
step = int(input("Enter the step value:"))
#Crate the generator
sequence_generator = number_sequence (start, end, step)
# Print the generated sequence of number
  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)! Produce a default sequence of numbers start from 0, ending at 10, with a step 1 if no value

Aim: To produce a default sequence of number start from 0 ending 10.

Algorithm:

1. Start Function:
   - Define the function my_generator(n) then takes a parameter
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 obj.
5. Iterate & Print Values:
   - For each value produced by generator object
     - Print values.

Program:

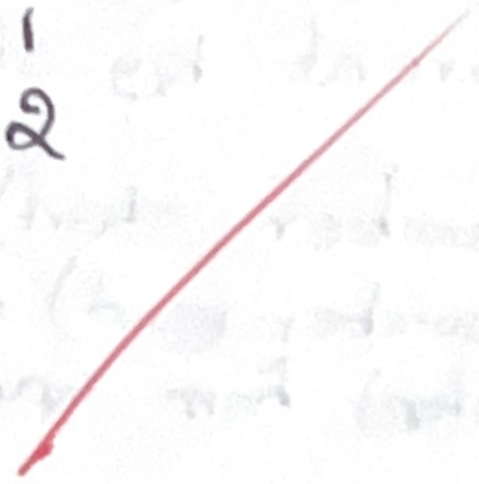
```
def my_generator(n):
  #initialize counter
  value =0
  # loop until counter is less than n
  while value < n:
  #produce the current value of counter.
    yield value
  # increment the counter.
    value += 1
#iterate over the generator object produced by my_generator
  for value in my_generator(3):
#print each value produced by generator:
    print(value)
```

Output

0

1

2



Output

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

hi, i am creted by a function possed as an argumet

Task 8.2

Imagine you are working on a messaging application that needs to format messages differently based

Aim:

Algorithm:

1. Create Decorators:
   - Define uppercase_decorator to convert the result of a function to uppercase
   - Define lower_case decorator to convert result 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
3. Define Greet Function:
   - Define greet function that:
     - Accepts a function as input.
     - Calls this function with the text "Hi, I am created by a function.

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
def greet(func):
    greeting = func("Hi, I am created by a function passed as argument")
    print(greeting)
greet(shout)
greet(whisper)
```

PERFORMANCE (5)
RESULT AND ANALYSIS (5)
VIVA VOCE (5)
RECORD (5)
TOTAL (20)
SIGN WITH DATE

Result: Thus the Python program to implement python generators and decorators was successfully executed & output verified.

PERFORMANCE (5)	5
RESULT AND ANALYSIS (3)	3
VIVA VOCE (3)	3
RECORD (4)	4
TOTAL (15)	15
SIGN WITH DATE	