Task-7 Implementation python generator and decoders.
8/9/25

Aim:- write a python program to implement python generator and decorators.

a. produce a sequence of number when provided with start, end and step values.

b. produce a default sequence of number starting from 0, ending 10, and with a step1 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 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 usr input.
5. Create generator object:
   - Create a generator object by calling number_sequence with user-provided values.
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.
```

Output:

```
Enter the starting number: 1
Enter the ending number: 50
Enter time step value: 5
 6
11
16
21
26
31
36
41
46.
```

```
start = int ("Enter the starting number:")
end = int (input("Enter the ending number:"))
# create the generator
sequence_generator = number_sequence (start, end, step)
# print the generator sequence of number.
for number is sequence_generator.
  print (number)
```

7.1(b) ~~program~~:- Produce a default sequence of number starting from 0, ending at 10, and with a step of 1 value provided.

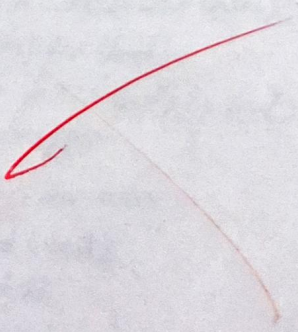
Algorithm:

1. Start function:
   Define the function my_generator (n) that takes parameter n,
2. Initialize counter:
   - Set value to 0.
3. generate values:
   - while value is less than n:
4. Create generator object:
   - call my_generator (11) to create generator object.
5. Iterate and print values:
   For each value produced by generator object:
   Print value

Program:

```
def my_generator (n):
# initilize counter
  value :0
# Loop until counter less than n.
  while value<n:
  # produce the current value of counter.
    yield value
  # increment the counter
  Value +=1
```

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



Task (7.2) Algorithm:-
8/9/25

Algorithm:

1. Create Decorators:-
   - Define uppercase-decorator to convert the result of a function to uppercase.
   - Define lowercase-decorator to convert result of a function to lowercase.
2. Define functions:
   - Define shout function to return the input text. Apply @upper-decorator to this function.
3. Define Greet function:
   - Define greet function that:
   - Accepts a function (func) as input:
   - calls this function with text "Hi, I am created by function passed as an argument.
   - Prints the result.
4. Execute program:
   - call greet(shout) to print greeting in uppercase.
   - call greet(whisper) to print greeting in lower case.

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
```

greet(shout)
greet(whisper)

VEL TECH - CSE	
EX NO.	1
PERFORMANCE (5)	5
RESULT AND ANALYSIS (5)	5
VIVA VOCE (5)	5
RECORD (5)	
TOTAL (20)	15
SIGN WITH DATE	

Result:

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

~~hi, I am created by function passed as an argument.~~

Thus, the python program to implement python generator generator and decorator was successfully executed and output was verified.