

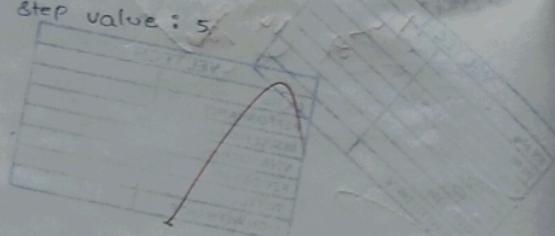
* Program:

```
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: "))  
  
sequence_generator = number_sequence(start, end, step)  
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
```



Date: 15/9/25

TASK-8 Implement Python generator and decorators-

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

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 no. (`start`) from user input.
- Read the ending no. (`end`) from user input.
- Read the step value (`step`) from user input.

5. Print Generated Sequence:

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

8.1(b) Program:

```
def my_generator(n):
    value=0
    while value < n:
        yield value
        value+=1
for value in my_generator(3):
    print(value)
```

Output:-

0
1
2



- b) Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no value.
- Algorithm:- [are provided]

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()` to create a generator object.

5. Iterate and print values:

- For each value produced by the generator object:
- Print value

NAME	ADDRESS	PHONE NUMBER
John Doe	123 Main St	555-1234
Jane Doe	456 Elm St	555-2345
Bob Smith	789 Oak St	555-3456
Susan Smith	210 Pine St	555-4567

8.1(b) 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")
    print(greeting)

```

Output:-

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

Hi, I am created by a function passed as an argument.

VELTECH	
EX No.	
PERFORMANCE (5)	B
RESULT AND ANALYSIS (5)	D
VIVA VOCE (5)	C
RECORD (5)	
TOTAL (20)	14
SIGN WITH DATE	

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.

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.

VELTECH	
EX No.	
PERFORMANCE (5)	
RESULT AND ANALYSIS (5)	
VIVA VOCE (5)	
RECORD (5)	
TOTAL (15)	
SIGN WITH DATE	

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