* 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:-

[illegible]

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:
     - ~~Ye~~ 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:
   yeild value
    value += 1
for value in my_generator (3):
 print(value)
```

Output:-

```
0
1
2
```

b) Produce a defult sequence of numbers Startiug 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 value.
```

8.1(b) Progr[illegible]

```
def [illegible] decorator (func):
    def wrapper (text):
        return fun(text).upper()
    return wrapper

def lowercase_decorator (func):
    def wrapper (text):
        return func(text).lower()
    return wrapper

@uppercase_decorater
def shout(text):
    return text

@lowercase_decorater
def whisper (text):
    return text

def greet (func)
    greeting = func("Hi, I am Created by a function")
    print (greeting)

greet(Shout)
greet (whisper)
```

VELTECH	
EX No.	8
PERFORMANCE (5)	05
RESULT AND ANALYSIS (5)	04
VIVA VOCE (5)	[illegible]
RECORD (5)	[illegible]
TOTAL (20)	19
SIGN WITH DATE	[illegible]

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 Imagine you are working on a messaging application that needs to format messages differently based on the users preferences. users Can choose to have their messages automatically Converted to uppercase (or) to lowercase.

Algorithm:-

1. Create Decorators:
   - Define upper_case 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_deorator 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 (3)	
VIVA VOCE (3)	
RECORD (4)	
TOTAL (15)	
SIGN WITH DATE	

RESULT:- Thus, the python to implement python generator and decorators was successfully executed and the output was verified.