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
```

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
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 from user input.
   - Read the ending number from user input.
   - Read the step value from user input.
5. Create Generator Object:
   - create a generator object by calling number_sequence
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 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)
```

Result:- Thus the python program for generating the sequence of numbers was successfully verified.

Output:- 0
1
2

8.7(b)

Aim:- To write the Python program my-generator using loop statements.

Algorithm:-

1. Start Function:
   - Define the function my-generator(n) that takes a parameter n.
2. Initialize Counter: Set values to 0.
3. Generate values: While value is less than n:
   - Yield current value
   - Increment by value 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.

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

Result:- Thus the python program my-generator using loop statements was successfully executed.

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

Aim:- To write a python program using functions they decorate by converting the textcase.

1. Create Decorators:
   - Define uppercase_decorator to convert the result of a function to uppercase
2. Define functions:
   - Define shout function to return the input text.
3. Defines greet function:
   - Accepts a function (func) as input.
   - Print the result.
4. Execute the program.
   - call greet to print text.

```
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 an argument.")
    print(greeting)
greet(shout)
greet(whisper)
```

VEL TECH	
Ex. No.	8
PERFORMANCE (5)	5
RESULT AND ANALYSIS	5
VIVA VOCE (3)	5
RECORD (4)	
TOTAL (15)	
SIGN WITH DATE	15

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