**Task 8.Implement python generator and decorators**

**Aim**:

Write a python program to Implement python generator and decorators

***8.1 Write a Python program that includes a generator function to produce a sequence of numbers. The generator should be able to:***

1. ***Produce a sequence of numbers when provided with start, end, and step values.***
2. ***Produce a default sequence of numbers starting from 0, ending at 10, and with a step of 1 if no values are provided.***

**Produce a sequence of numbers when provided with start, end, and step values.**

**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 number (start) from user input.
   * Read the ending number (end) from user input.
   * Read the step value (step) from user input.
5. **Create Generator Object:**
   * Create a generator object by calling number\_sequence(start, end, step) with user-provided values.
6. **Print Generated Sequence:**
   * Iterate over the values produced by the generator object.
   * Print each value.

**8.1. 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)

**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

**Produce a default sequence of numbers starting 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)Program:**

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

# 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

***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 (for emphasis) or to lowercase (for a softer tone). You are provided with two decorators: uppercase\_decorator and lowercase\_decorator. These decorators modify the behavior of the functions they decorate by converting the text to uppercase or lowercase, respectively. Write a program to implement it.***

**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:**
   * Call greet(shout) to print the greeting in uppercase.
   * Call greet(whisper) to print the greeting in lowercase.

**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 an argument.")

print(greeting)

greet(shout)

greet(whisper)

**Output:**

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

hi, i am created by a function passed as an argument.

**Result:**

Thus the python program to Implement python generator and decorators was successfully executed and the output was verified.