**Task 8.Implement python generator and decorators CO1 and CO2-K3**

**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 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

***8.2 Write a Python program that uses a generator function to produce all prime numbers up to a given limit n. A prime number is a number greater than 1 that has no divisors other than 1 and itself.***

**Algorithm:**

1. **Define Generator Function**

Define prime\_generator(n) that generates prime numbers up to n.

1. **Loop through Numbers**

Iterate from 2 to n (inclusive).

1. **Check for Primality**

For each number num, check divisibility from 2 to √num.

If divisible by any, it’s not a prime → break.

If no divisors found, it’s a prime → yield num.

1. **Create Generator Object**

Call prime\_generator(n) with user-provided value n.

1. **Print Generated Primes**

Iterate over the values produced by the generator.

Print each prime number.

**8.2. Program**:

def prime\_generator(n):

# Generate prime numbers up to n

for num in range(2, n+1):

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

break

else:

yield num

# Take user input

n = int(input("Enter the upper limit to generate primes: "))

# Create the generator

prime\_gen = prime\_generator(n)

# Print the generated prime numbers

print(f"Prime numbers up to {n}:")

for p in prime\_gen:

print(p)

**Output:**

Enter the upper limit to generate primes: 20

Prime numbers up to 20:

2

3

5

7

11

13

17

19

***8.3 Write a Python program that uses a generator function to produce all prime numbers up to a given limit n, and a decorator to log each prime number as it is generated.***

**Algorithm:**

1. **Define Decorator Function**

Create a decorator log\_prime(func) that takes a generator function as input.

Inside it, define a wrapper function that:

* 1. Calls the original generator function.
  2. For each prime yielded, prints a log message (Prime generated: X).
  3. Yields the prime back to the caller.

1. **Define Prime Generator**

Define prime\_generator(n) to generate prime numbers up to n.

For each number, check divisibility from 2 to √num.

Yield the number if no divisor is found.

Decorate it with @log\_prime so that logging is added automatically.

1. **Call the Generator**

Run prime\_generator(n) for a given limit (e.g., 20).

Iterate through the generator.

1. **Print Logs**

Observe that each prime number is logged automatically by the decorator.

**8.3. Program**:

def log\_prime(func):

def wrapper(n):

for prime in func(n):

print(f"Prime generated: {prime}") # log each prime

yield prime

return wrapper

@log\_prime

def prime\_generator(n):

for num in range(2, n+1):

for i in range(2, int(num\*\*0.5)+1):

if num % i == 0:

break

else:

yield num

# Generate primes up to 20

for p in prime\_generator(20):

pass

**Output:**

Prime generated: 2

Prime generated: 3

Prime generated: 5

Prime generated: 7

Prime generated: 11

Prime generated: 13

Prime generated: 17

Prime generated: 19

***8.4.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.