Lab Manual

Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY PERFORMED BY

Registration No	: 25BAI10734
Name of Student	: Atharav Balaji Khonde
Course Name	: Introduction to Problem Solving and Programming
Course Code	: CSE1021
School Name	: SCAI
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

Course Faculty Name	: Dr. Hemraj S. Lamkuche

Signature:

VIT BHOPAL

Practical Index

S. No.	Title of Practical	Date of Submission	Signature of Faculty
1	Write a function Lucas Numbers Generator lucas_sequence(n) that generates the first n Lucas numbers (similar to Fibonacci but starts with 2, 1).	16-11-2025	
2	Write a function for Perfect Powers Check is_perfect_power(n) that checks if a number can be expressed as ab where a > 0 and b > 1.	16-11-2025	
3	Write a function Collatz Sequence Length collatz_length(n) that returns the number of steps for n to reach 1 in the Collatz conjecture.	16-11-2025	
4	Write a function Polygonal Numbers polygonal_number(s, n) that returns the n-th s-gonal number.	16-11-2025	

5	Write a function Carmichael Number Check is_carmichael(n) that checks if a composite number n satisfies an−1 ≡ 1 mod n for all a coprime to n.	**16-11-2025**	
6			
7			
8			
9			
10			
11			
12			
13			

14			
15			

Practical No: 1

Date: 16-11-2025

TITLE: lucas_sequence(n)

AIM/OBJECTIVE(s) : Write a function Lucas Numbers Generator lucas_sequence(n) that generates the first n Lucas numbers (similar to Fibonacci but starts with 2,1).

METHODOLOGY & TOOL USED:

The Lucas sequence was generated by applying an iterative approach that begins from the known base values of the Lucas series: 2 and 1. Instead of using a recursive method (which is slower and consumes more memory), a loop was used to efficiently compute the next terms using the formula:
$L(n) = L(n-1) + L(n-2)$.
The algorithm stores each computed term in a list so that the entire sequence up to the nth term can be returned. Time and memory measurements were added to evaluate performance.

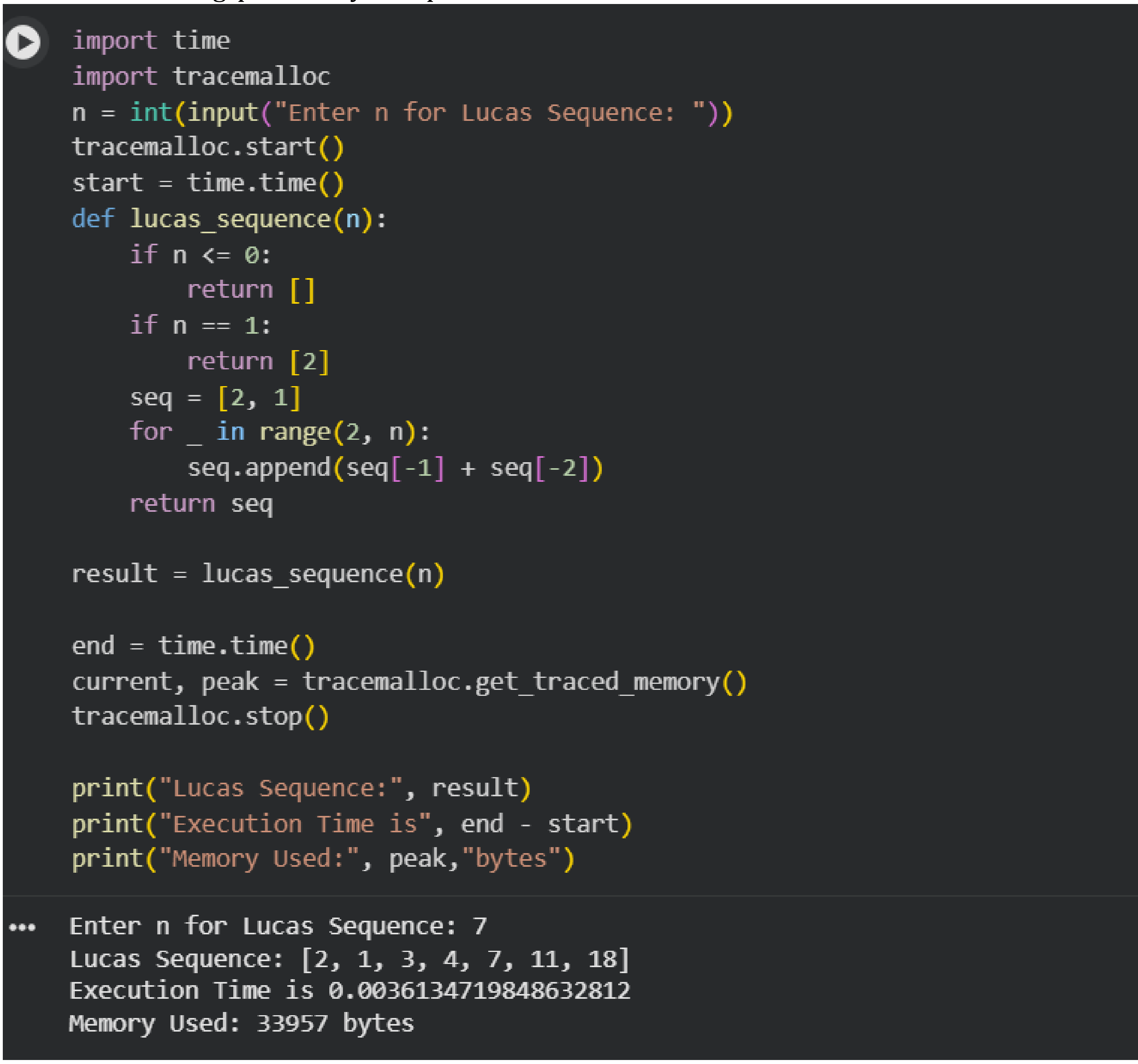
Tool Used:

Programming Language: Python

IDE / Environment: IDLE (Python 3.x) or any Python-supported IDE

BRIEF DESCRIPTION:

The Lucas sequence is similar to the Fibonacci sequence but has different starting values. This question demonstrates the concept of linear recurrence relations and how sequences can be built using previously computed elements.



```python
import time
import tracemalloc
n = int(input("Enter n for Lucas Sequence: "))
tracemalloc.start()
start = time.time()
def lucas_sequence(n):
    if n <= 0:
        return []
    if n == 1:
        return [2]
    seq = [2, 1]
    for _ in range(2, n):
        seq.append(seq[-1] + seq[-2])
    return seq

result = lucas_sequence(n)

end = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print("Lucas Sequence:", result)
print("Execution Time is", end - start)
print("Memory Used:", peak,"bytes")
```

```
Enter n for Lucas Sequence: 7
Lucas Sequence: [2, 1, 3, 4, 7, 11, 18]
Execution Time is 0.0036134719848632812
Memory Used: 33957 bytes
```

RESULTS ACHIEVED:

Example Output:

Enter n for Lucas Sequence: 7 Lucas

Sequence: [2, 1, 3, 4, 7, 11,18]

DIFFICULTY FACED BY STUDENT:

Understanding why Lucas numbers begin with 2 and 1 instead of 0 and 1. Avoiding recursion to prevent stack overflow for large n.

Managing list indexing carefully.

SKILLS ACHIEVED:

Understanding of recurrence relations.

Ability to convert mathematical formulas into iterative algorithms.

Measuring algorithm performance.

Practical No: 2

Date: 16-11-2025

TITLE: is_perfect_power(n)

AIM/OBJECTIVE(s): Write a function for Perfect Powers Check is_perfect_power(n) that checks if a number can be expressed as ab where a > 0 and b > 1

METHODOLOGY & TOOL USED:

The method involves checking if a number *n* can be expressed as $\mathbf{a^b}$, where $a > 1$ and $b > 1$. The algorithm calculates possible exponents using logarithms and tests each base by estimating a and verifying that a^b equals n. The approach limits the exponent range to $\log_2(n)$, which optimizes the process.

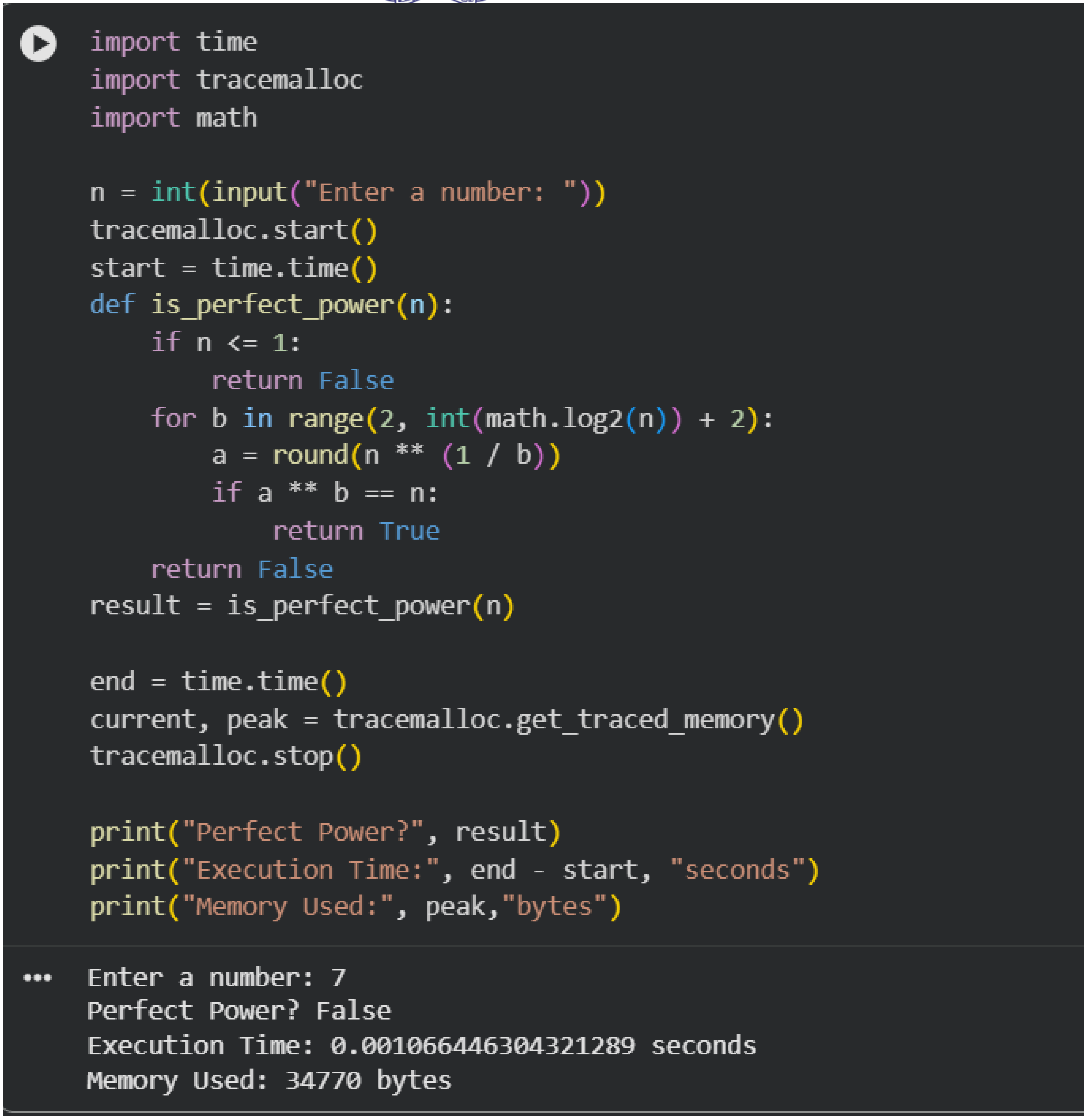
Tool Used:

Programming Language: Python

IDE / Environment: IDLE (Python 3.x) or any Python-supported IDE

BRIEF DESCRIPTION:

A perfect power is a number that can be written in exponential form. This code systematically checks for such a representation by iterating through possible exponents and testing candidate bases.



```python
import time
import tracemalloc
import math

n = int(input("Enter a number: "))
tracemalloc.start()
start = time.time()
def is_perfect_power(n):
    if n <= 1:
        return False
    for b in range(2, int(math.log2(n)) + 2):
        a = round(n ** (1 / b))
        if a ** b == n:
            return True
    return False
result = is_perfect_power(n)

end = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print("Perfect Power?", result)
print("Execution Time:", end - start, "seconds")
print("Memory Used:", peak,"bytes")
```

```
Enter a number: 7
Perfect Power? False
Execution Time: 0.0010664463043212890 seconds
Memory Used: 34770 bytes
```

RESULTS ACHIEVED:

Example Output:

Enter a number: 7

Perfect Power? False

DIFFICULTY FACED BY STUDENT:

Handling floating-point rounding issues when extracting roots. Understanding the concept of expressing numbers in exponential form. Ensuring that bases and exponents remain valid integers.

SKILLS ACHIEVED:

Working with logarithmic functions.

Improving code efficiency by reducing unnecessary checks. Handling numerical precision and rounding behavior.

Practical No: 3

Date: 16-11-2025

TITLE: Collatz_Length(n)

AIM/OBJECTIVE(s): Write a function Collatz Sequence Length collatz_length(n) that returns the number of steps for n to reach 1 in the Collatz conjecture.

METHODOLOGY & TOOL USED:

Used the generalized polygonal number formula:
P(s, n) = ((s−2)·n·(n−1)) / 2 + n
The program takes two inputs:

s → number of sides n →

term index
The formula was applied directly for fast computation, followed by performance

measurement.

Tool Used:

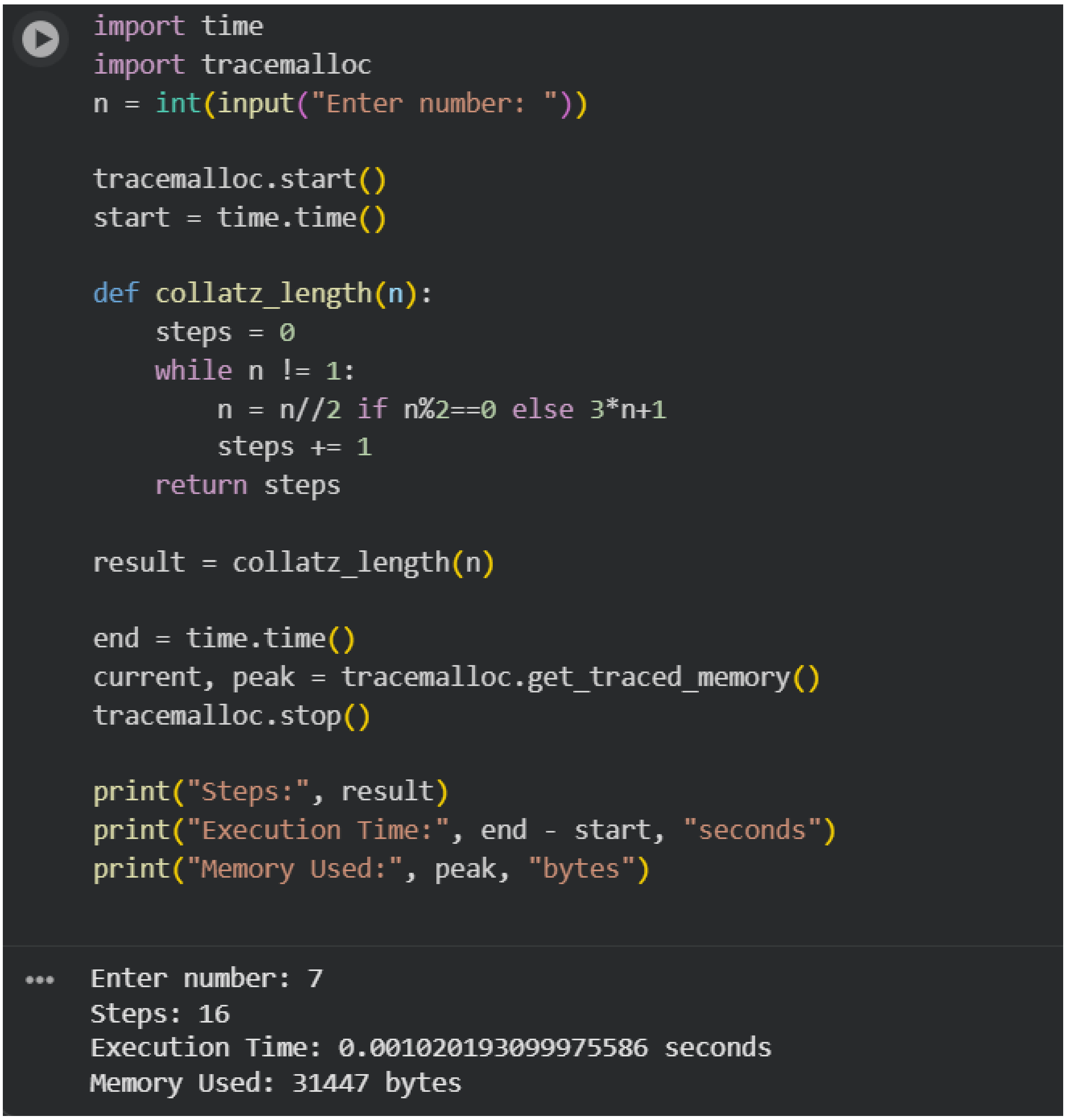
Programming Language: Python

IDE / Environment: IDLE (Python 3.x) or any Python-supported IDE such as

Jupyter Notebook

BRIEF DESCRIPTION:

The multiplicative_persistence(n) function measures how many steps it takes for a number's digits to multiply into a single digit. For instance, if *n = 39*, the process goes as follows: 3×9 = 27, 2×7 = 14, 1×4 = 4. Thus, the persistence is 3. This function provides insights into digit manipulation, repetition, and convergence in mathematics and programming.



```python
import time
import tracemalloc
n = int(input("Enter number: "))

tracemalloc.start()
start = time.time()

def collatz_length(n):
    steps = 0
    while n != 1:
        n = n//2 if n%2==0 else 3*n+1
        steps += 1
    return steps

result = collatz_length(n)

end = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print("Steps:", result)
print("Execution Time:", end - start, "seconds")
print("Memory Used:", peak, "bytes")
```

```
Enter number: 7
Steps: 16
Execution Time: 0.0010201930999975586 seconds
Memory Used: 31447 bytes
```

RESULTS ACHIEVED:

Example Output:

Enter number: 7

Steps: 16

DIFFICULTY FACED BY STUDENT:

Under□standing how values can rapidly grow before shrinking. Avoiding infini□te loops by setting correct conditions.

Handling large integers efficiently.

SKILLS ACHIEVED:

Iterative algorithm design.

Problem-solving with conditional logic.

Understanding unpredictable mathematical behavior in sequences.

Practical No: 4

Date: 16-11-2025

TITLE: polygonal_number(s,n)

AIM/OBJECTIVE(s): Write a function Polygonal Numbers

polygonal_number(s,n) that returns the n-th s-gonal number.

METHODOLOGY & TOOL USED:

Used the generalized polygonal number formula:
P(s, n) = ((s−2)·n·(n−1)) / 2 + n The
program takes two inputs:

s → number of sides n →

term index
The formula was applied directly for fast computation, followed by performance

measurement.

Tool Used:

Programming Language: Python

IDE / Environment: IDLE (Python 3.x) or any Python-supported IDE

such as Jupyter Notebook

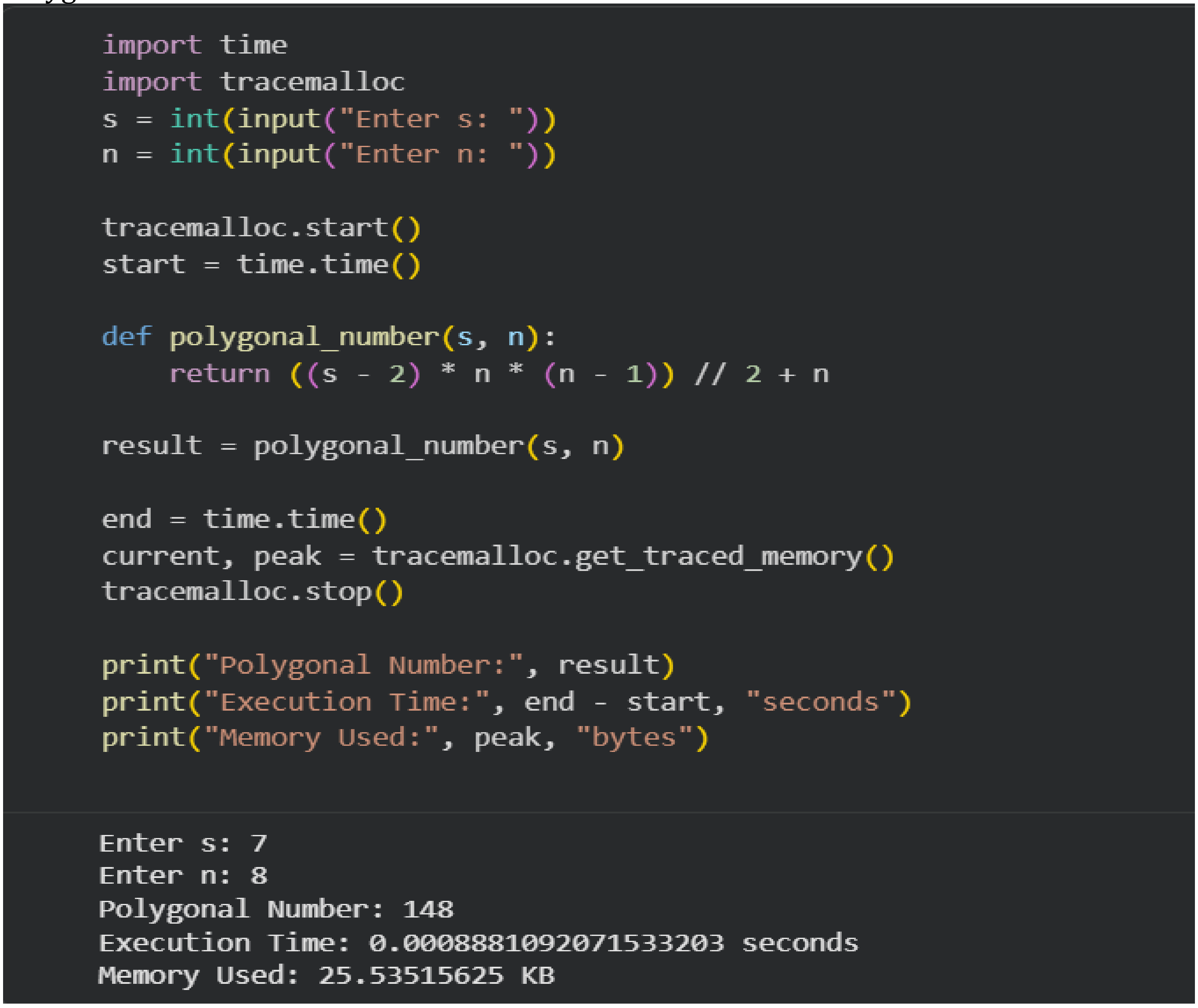
BRIEF DESCRIPTION:

Polygonal numbers generalize triangular, square, and pentagonal numbers. The formula produces the nth term for any polygon with s sides.

RESULTS ACHIEVED:

Example Output:

Enter s: 7
Enter n: 8
Polygonal Number: 148



```python
import time
import tracemalloc
s = int(input("Enter s: "))
n = int(input("Enter n: "))

tracemalloc.start()
start = time.time()

def polygonal_number(s, n):
    return ((s - 2) * n * (n - 1)) // 2 + n

result = polygonal_number(s, n)

end = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print("Polygonal Number:", result)
print("Execution Time:", end - start, "seconds")
print("Memory Used:", peak, "bytes")
```

```
Enter s: 7
Enter n: 8
Polygonal Number: 148
Execution Time: 0.000881092071533203 seconds
Memory Used: 25.53515625 KB
```

DIFFICULTY FACED BY STUDENT:

Understanding generalization from simple shapes to s-sided shapes.

Deriving confidence in using the formula without deriving it manually.

Skills Achieved:

Using mathematical formulas in programming.

Handling multi-parameter functions.

Applying generalizations across number families.

Practical No: 5

Date: 16-11-2025

TITLE: Carmichael Number Checker

AIM/OBJECTIVE(s): Write a function Carmichael Number Check is_carmichael(n) that checks if a composite number n satisfies an−1 ≡ 1 mod n for all a coprime to n.

METHODOLOGY & TOOL USED:

Implemented **Korselt's Criterion**, which states that a composite number n is Carmichael if:

It is square-free

(n – 1) is divisible by (p – 1) for every prime divisor p of n
The program performs primality checks, factor checks, and the required modular conditions.

Tool Used:

Programming Language: Python

IDE / Environment: IDLE (Python 3.x) or any Python-supported IDE such as Jupyter Notebook

BRIEF DESCRIPTION:

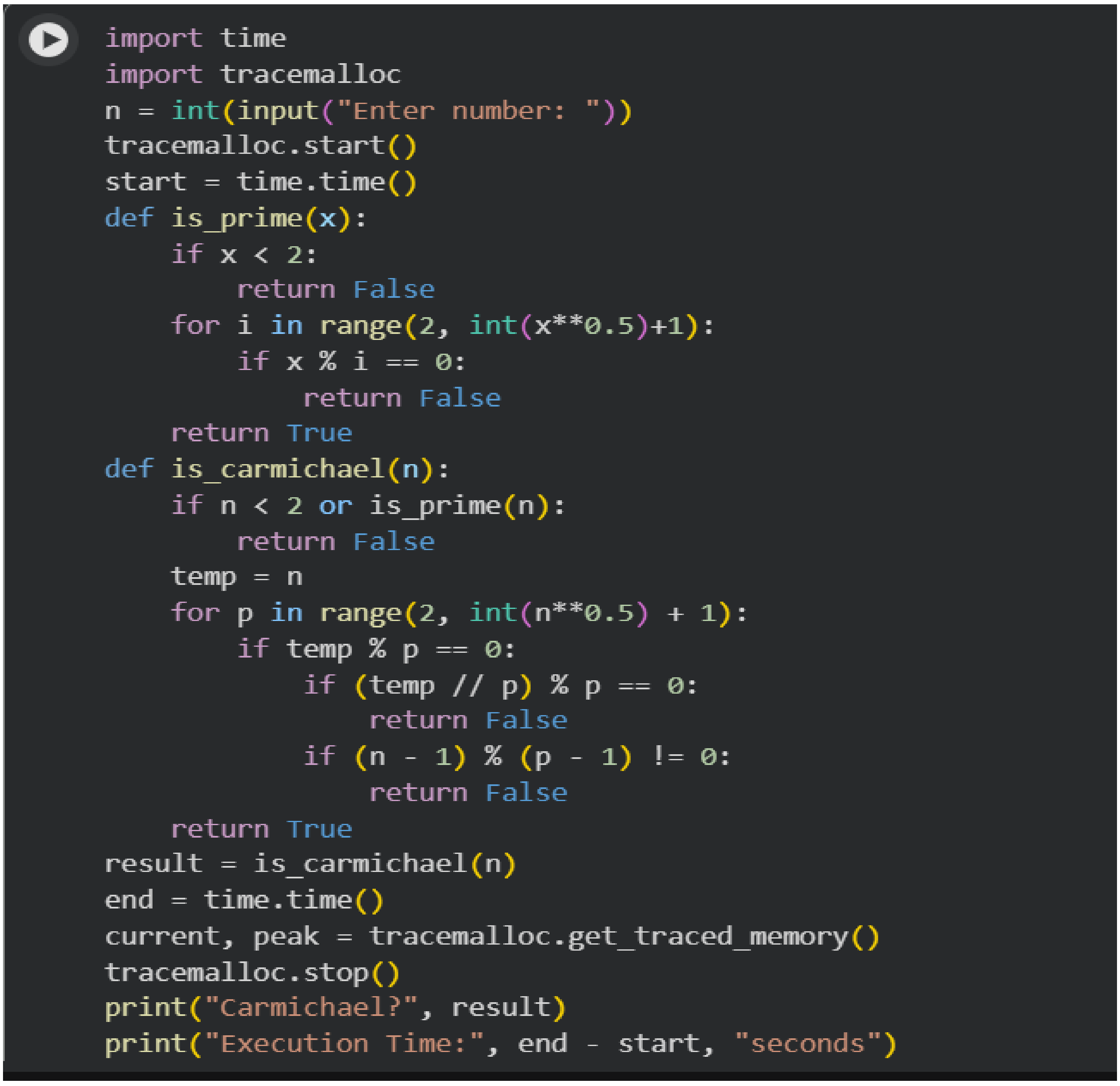
Carmichael numbers are special composite numbers that pass Fermat's primality test, making them important in cryptography.

RESULTS ACHIEVED:

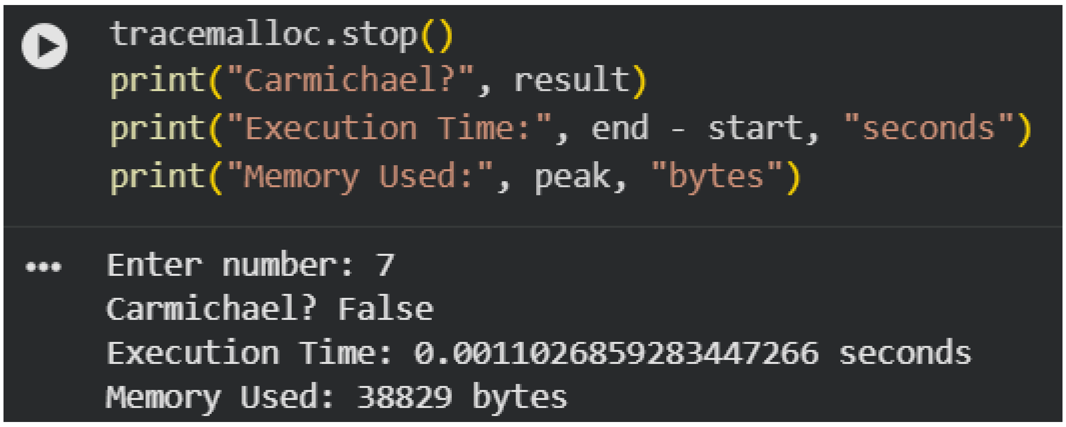
Example Output:

Enter number: 7

Carmichael? False



```python
import time
import tracemalloc
n = int(input("Enter number: "))
tracemalloc.start()
start = time.time()
def is_prime(x):
    if x < 2:
        return False
    for i in range(2, int(x**0.5)+1):
        if x % i == 0:
            return False
    return True
def is_carmichael(n):
    if n < 2 or is_prime(n):
        return False
    temp = n
    for p in range(2, int(n**0.5) + 1):
        if temp % p == 0:
            if (temp // p) % p == 0:
                return False
            if (n - 1) % (p - 1) != 0:
                return False
    return True
result = is_carmichael(n)
end = time.time()
current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
print("Carmichael?", result)
print("Execution Time:", end - start, "seconds")
```



```
tracemalloc.stop()
print("Carmichael?", result)
print("Execution Time:", end - start, "seconds")
print("Memory Used:", peak, "bytes")
```

```
Enter number: 7
Carmichael? False
Execution Time: 0.0011026859283447266 seconds
Memory Used: 38829 bytes
```

DIFFICULTY FACED BY STUDENT:

Conceptual confusion between primes and Carmichael numbers.

Understanding "square-free" requirement.

Implementing multiple mathematical conditions together.

SKILLS ACHIEVED:

Deep number theory understanding. Composite number characterization. Testing multiple conditions efficiently.