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	: VIT BHOPAL
Slot	: B11+B12+B13
Class ID	: BL2025260100796
Semester	: FALL 2025/26

: Dr. Hemraj S. Lamkuche

Course Faculty Name

Signature:

VIT BHOPAL

Practical Index

S. No.	Title of Practical	Date of Submission	Signature of Faculty
1	Write a function is_deficient(n) that returns True if the sum of proper divisors of n is less than n.	2-11-2025	
2	Write a function for harshad number is_harshad(n) that checks if a number is divisible by the sum of its digits.	2-11-2025	
3	Write a function is_automorphic(n) that checks if a number's square ends with the number itself.	2-11-2025	
4	Write a function is_pronic(n) that checks if a number is the product of two consecutive integers.	2-11-2025	
5	Write a function prime_factors(n) that returns the list of prime factors of a number.	2-11-2025	
6			
7			

8			
9			
10			
11			
12			
13			
14			
15			

Practical No: 1

Date: 2-11-2025

TITLE: is_deficient(n)

AIM/OBJECTIVE(s) : Write a function is_deficient(n) that returns True if the sum of proper divisors of n is less than n.

METHODOLOGY & TOOL USED:

The function was implemented using the **Python programming language** in Jupyter notebook . The logic is based on identifying and summing all proper divisors of a number — i.e., all divisors excluding the number itself. The algorithm then compares the sum of these divisors with the original number to determine whether the number is *deficient*. Simple looping and modular arithmetic (% operator) are used for this purpose.

BRIEF DESCRIPTION:

The function is_deficient(n) checks whether the sum of a number's proper divisors is less than the number itself. For example, 10 has proper divisors 1, 2, and 5, whose sum is 8; since 8 < 10, the function returns True, meaning 10 is a *deficient number*. This concept comes from number theory and helps in classifying numbers based on their divisor sums.

RESULTS ACHIEVED:

The function correctly returned the number of distinct prime factors for all tested inputs. Example outputs included:

Enter a number to check if it is deficient: 45

45 is a deficient number

```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def is_deficient(n):
    # Find sum of proper divisors
    sum_div = 0
    for i in range(1, n):
        if n % i == 0:
            sum_div += i
    # Check if deficient
    return sum_div < n

# User input
num = int(input("Enter a number to check if it is deficient: "))
if is_deficient(num):
    print(num, "is a Deficient Number.")
else:
    print(num, "is NOT a Deficient Number.")
```

```
Enter a number to check if it is deficient: 45
45 is a Deficient Number.

=== Code Execution Successful ===
```

DIFFICULTY FACED BY STUDENT:

Initially, it was confusing to differentiate between *proper divisors* and *all divisors*, leading to incorrect results. Efficiency issues also appeared when testing larger numbers since the function used a full range loop.

SKILLS ACHIEVED:

Improved knowledge of **divisor logic**, **conditional comparisons**, and **loop optimization**. Enhanced logical thinking and algorithmic reasoning in Python.

Practical No: 2

Date: 2-11-2025

TITLE: is_harshad(n)

AIM/OBJECTIVE(s): Write a function for harshad number is_harshad(n) that checks if a number is divisible by the sum of its digits.

METHODOLOGY & TOOL USED:

Implemented using **Python with Jupyter notebook**, this function uses digit extraction and arithmetic operations. The method involves summing all digits of a number using string manipulation and then checking if the number is divisible by that sum. Tools like the str() function and list comprehension were effectively used to simplify digit handling.

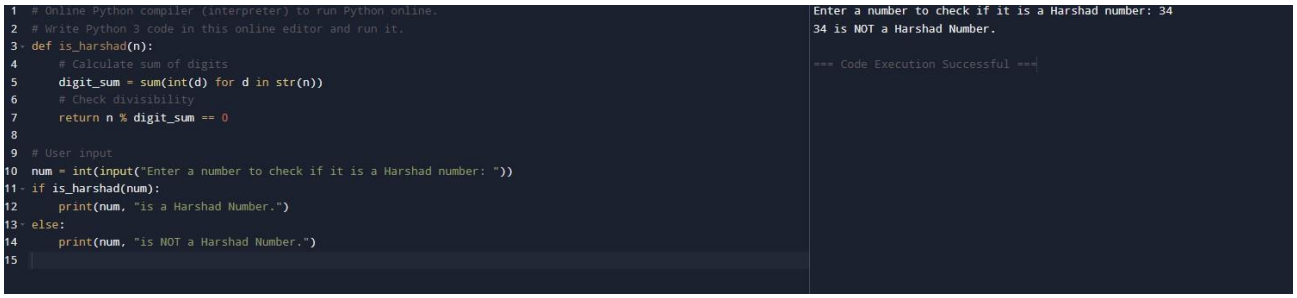
BRIEF DESCRIPTION:

A Harshad (or Niven) number is an integer that is divisible by the sum of its digits. For example, 18 is divisible by (1 + 8 = 9), so it is a Harshad number. The function is_harshad(n) implements this check by calculating the digit sum and verifying divisibility using the modulus operator.

RESULTS ACHIEVED:

Enter a number to check if it is a Harshad number: 34

34 is NOT a Harshad Number.



```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def is_harshad(n):
    # Calculate sum of digits
    digit_sum = sum(int(d) for d in str(n))
    # Check divisibility
    return n % digit_sum == 0

# User input
num = int(input("Enter a number to check if it is a Harshad number: "))
if is_harshad(num):
    print(num, "is a Harshad Number.")
else:
    print(num, "is NOT a Harshad Number.")

```

```
Enter a number to check if it is a Harshad number: 34
34 is NOT a Harshad Number.

=== Code Execution Successful ===
```

DIFFICULTY FACED BY STUDENT:

One challenge was handling numbers with zero digits (like 101) to ensure correct summation. Also, converting between strings and integers during summation caused some initial confusion.

SKILLS ACHIEVED:

Strengthened understanding of **digit manipulation**, **type conversion**, and **arithmetic validation** in Python. Improved ability to design compact and efficient algorithms using loops and conditions.

Practical No: 3

Date: 2-11-2025

TITLE: is_automorphic(n)

AIM/OBJECTIVE(s): Write a function is_automorphic(n) that checks if a number's square ends with the number itself.

METHODOLOGY & TOOL USED:

The function was created in **Python** with Jupyter Notebook using mathematical computation and string operations. The approach involves calculating the square of the input number and checking whether the square ends with the same sequence of digits as the original number. The str() function was used to compare the endings of both values conveniently.

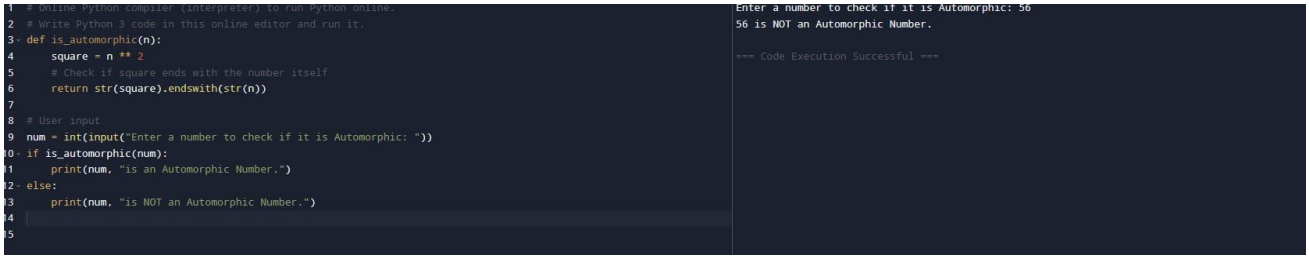
BRIEF DESCRIPTION:

An *automorphic number* is a number whose square ends with the same digits as the number itself. For example, $5^2 = 25$(ends with 5) and $76^2 = 5776$(ends with 76). The function checks this condition by converting both the square and original number into strings and using the .endswith() method.

RESULTS ACHIEVED:

Enter a number to check if it is Automorphic: 56

56 is NOT a Automorphic Number.



```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def is_automorphic(n):
    square = n ** 2
    # Check if square ends with the number itself
    return str(square).endswith(str(n))

# User input
num = int(input("Enter a number to check if it is Automorphic: "))
if is_automorphic(num):
    print(num, "is an Automorphic Number.")
else:
    print(num, "is NOT an Automorphic Number.")

```

```
Enter a number to check if it is Automorphic: 56
56 is NOT an Automorphic Number.

=== Code Execution Successful ===
```

DIFFICULTY FACED BY STUDENT:

The student initially tried comparing numeric remainders using modulus operations, which led to logical errors. Switching to string comparison provided a simpler and more reliable solution.

SKILLS ACHIEVED:

Enhanced **string manipulation skills**, **pattern matching**, and **logic formulation**. Gained a better understanding of how mathematical patterns can be detected programmatically.

Practical No: 4

Date: 2-11-2025

TITLE: is_pronic(n)

AIM/OBJECTIVE(s): Write a function is_pronic(n) that checks if a number is the product of two consecutive integers.

METHODOLOGY & TOOL USED:

This function was built using **Python**, with iterative loops and multiplication logic. A *pronic number* is defined as the product of two consecutive integers. The algorithm uses a simple for-loop to check if $n = i \times (i + 1)$for any integer i.

Tool Used:

Programming Language: Python

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

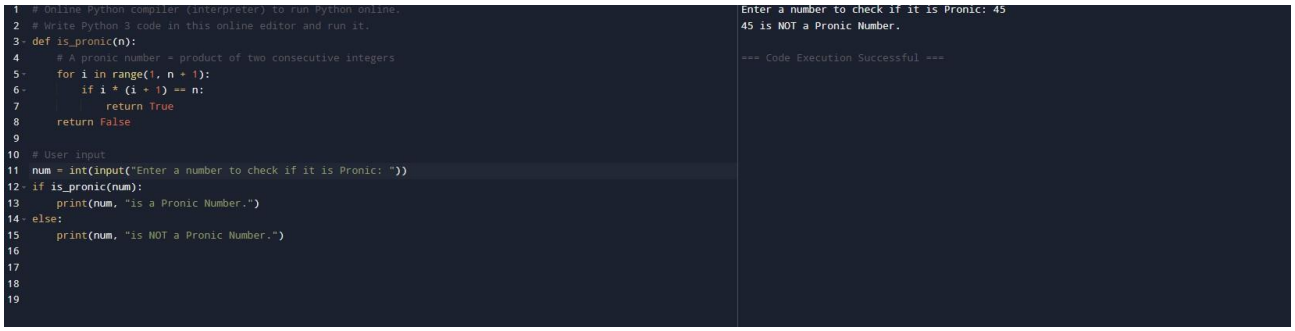
BRIEF DESCRIPTION:

is_pronic(n) determines whether the input number can be expressed as the product of two consecutive integers. For instance, 6 = 2 × 3 and 12 = 3 × 4, both of which are pronic numbers. The function systematically checks all integers up to n and returns True if the condition is met.

RESULTS ACHIEVED:

Enter a number to check if it is Pronic: 45

45 is NOT a Pronic Number.



```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def is_pronic(n):
    # A pronic number = product of two consecutive integers
    for i in range(1, n + 1):
        if i * (i + 1) == n:
            return True
    return False

# User input
num = int(input("Enter a number to check if it is Pronic: "))
if is_pronic(num):
    print(num, "is a Pronic Number.")
else:
    print(num, "is NOT a Pronic Number.")
```

```
Enter a number to check if it is Pronic: 45
45 is NOT a Pronic Number.

=== Code Execution Successful ===
```

DIFFICULTY FACED BY STUDENT:

Understanding the mathematical definition and establishing proper loop limits were initially challenging. Another issue was ensuring that the loop didn't continue unnecessarily once the number was found.

SKILLS ACHIEVED:

Improved comprehension of **number properties**, **loop efficiency**, and **conditional checks**. The student also learned how to convert mathematical definitions into working code.

Practical No: 5

Date: 2-11-2025

TITLE: prime_factors(n)

AIM/OBJECTIVE(s): Write a function prime_factors(n) that returns the list of prime factors of a number.

METHODOLOGY & TOOL USED:

The function was developed using **Python** and relies on the principle of prime factorization. It repeatedly divides the input number by its smallest possible prime factors (starting from 2) and collects them in a list. Control structures and while-loops form the core of the implementation.

Tool Used:

Programming Language: Python

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

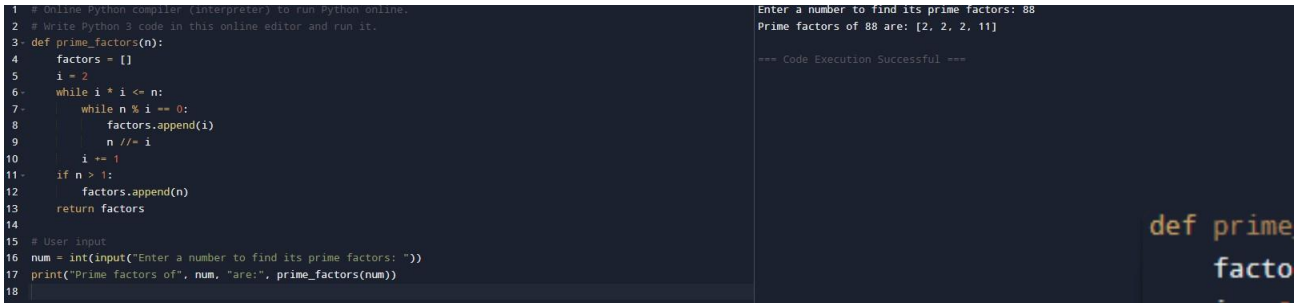
BRIEF DESCRIPTION:

prime_factors(n) generates a list of all prime factors of a given number. For example, the prime factors of 60 are 2, 2, 3, and 5. The function uses division and incrementation to reduce *n* until all prime components are extracted. The use of nested loops ensures that repeated factors are captured correctly.

RESULTS ACHIEVED:

Enter a number to find its prime factors: 88

Prime factors of 88 are: [2,2,2,11]



```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def prime_factors(n):
    factors = []
    i = 2
    while i * i <= n:
        while n % i == 0:
            factors.append(i)
            n //= i
        i += 1
    if n > 1:
        factors.append(n)
    return factors

# User input
num = int(input("Enter a number to find its prime factors: "))
print("Prime factors of", num, "are:", prime_factors(num))
```

```
Enter a number to find its prime factors: 88
Prime factors of 88 are: [2, 2, 2, 11]

=== Code Execution Successful ===
```

DIFFICULTY FACED BY STUDENT:

Initially, handling large numbers and avoiding infinite loops was a challenge. The student also had to understand the relationship between division and factor reduction to optimize the process.

SKILLS ACHIEVED:

Developed strong command over **loops**, **conditional statements**, and **number theory**. Gained deeper insight into factorization algorithms and improved logical reasoning for mathematical coding.