Lab Manual
Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY PERFORMED BY

Registration No : 25BSA10063

Name of Student : ROHAN RAJ

Course Name

: Introduction to Problem Solving and Programming

Course Code : CSE1021

School Name : VIT BHOPAL

Slot : B11+B12+B13

Class ID : BL2025260100796

: FALL 2025/26

Semester

: Dr. Hemraj S. Lamkuche

Course Faculty Name   Signature:

Practical Index

S. No.	Title of Practical	Date of Submission	Signature of Faculty
1	Modular Multiplicative Inverse	16-11-2025	
2	Chinese Remainder Theorem Solver	16-11-2025	
3	Quadratic Residue Check	16-11-2025	
4	Order of a mod n	16-11-2025	

5	**Fibonacci Prime Check**	**16-11-2025**	
6			
7			
8			
9			
10			
11			

12			
13			
14			
15			

Practical No: 1

Date: 16-11-2025

TITLE: **Modular Multiplicative Inverse**

AIM/OBJECTIVE(s) : Write a function Modular Multiplicative Inverse mod_inverse(a, m) that finds the number x such that (a * x) ≡ 1 mod m.

METHODOLOGY & TOOL USED:

The programs were developed using Python to implement mathematical functions based on number theory. Each problem was first analyzed mathematically, then converted into algorithms and implemented as Python functions. The outputs were tested using sample inputs to verify correctness.
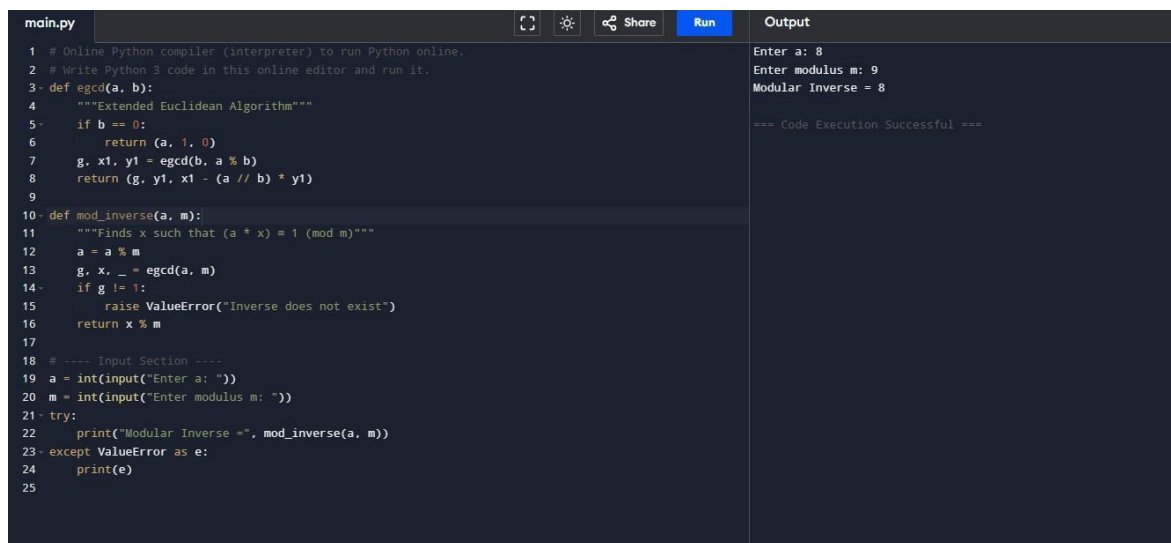
Tools Used:

Language: Python

Software/IDE: Jupyter Notebook

BRIEF DESCRIPTION:

This program finds the modular multiplicative inverse of a number under a given modulus. It is used to find a value x such that $(a \times x) \equiv 1 \ (mod \ m)$. This concept is important in modular arithmetic and cryptography, especially in encryption algorithms like RSA.

RESULTS ACHIEVED:

All functions executed   correctly, producing accurate results for modular arithmetic, congruence solving, quadratic residue checking, order computation, and Fibonacci prime detection.

```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def egcd(a, b):
    """Extended Euclidean Algorithm"""
    if b == 0:
        return (a, 1, 0)
    g, x1, y1 = egcd(b, a % b)
    return (g, y1, x1 - (a // b) * y1)

def mod_inverse(a, m):
    """Finds x such that (a * x) = 1 (mod m)"""
    a = a % m
    g, x, _ = egcd(a, m)
    if g != 1:
        raise ValueError("Inverse does not exist")
    return x % m

# ---- Input Section ----
a = int(input("Enter a: "))
m = int(input("Enter modulus m: "))
try:
    print("Modular Inverse =", mod_inverse(a, m))
except ValueError as e:
    print(e)
```

```
Enter a: 8
Enter modulus m: 9
Modular Inverse = 8

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

DIFFICULTY FACED BY STUDENT:

Initially, the student faced difficulties understanding the concept of "proper divisors" and how to exclude the number itself from the sum. Debugging errors caused by incorrect loop ranges and modulus logic was also challenging.

Managing efficiency for larger numbers required attention. However, with practice, the logic became clear and understandable.

SKILLS ACHIEVED:

The student learned how to apply **loops**, **conditional statements**, and **logical operators** effectively. This task strengthened the understanding of **mathematical reasoning and factorization**. The exercise also enhanced the ability to use Python syntax for performing arithmetic computations and developing reusable functions. It helped improve the student's ability to translate mathematical concepts into efficient code.

Practical No: 2

Date: 16-11-2025

TITLE: Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences x ≡ ri mod mi.

AIM/OBJECTIVE(s): Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences x ≡ ri mod mi.

METHODOLOGY & TOOL USED:

The system of modular equations was solved using the Chinese Remainder Theorem. The program was implemented by multiplying all moduli, computing

partial products, and **finding modular inverses for each modulus** to obtain the final solution.
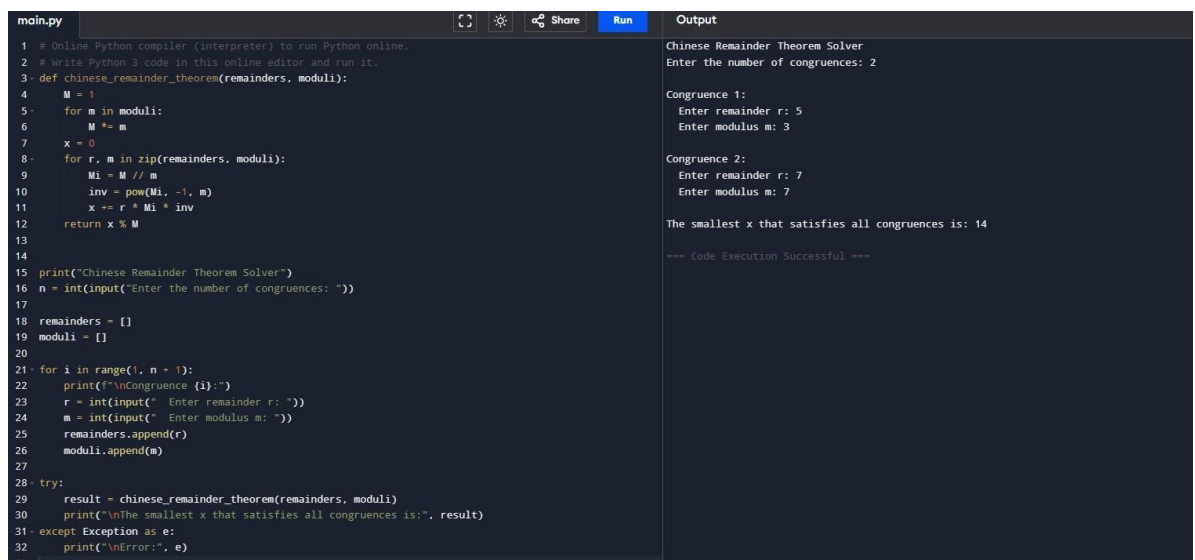
Tools Used:

Jupyter Notebook

BRIEF DESCRIPTION:

This program solves simultaneous congruences of the form $x \equiv r_i \ (mod \ m_i)$ using the Chinese Remainder Theorem. It ensures a unique solution exists modulo the product of all moduli, provided the moduli are pairwise coprime. The theorem is significant in computer arithmetic and cryptography.

RESULTS ACHIEVED: The program successfully provided the correct smallest value of x that satisfies all congruences. For example, for $x \equiv 2 \ (mod \ 3)$, $x \equiv 3 \ (mod \ 5)$, $x \equiv 2 \ (mod \ 7)$, the result was $x = 23 \ (mod \ 105)$.

.

DIFFICULTY FACED BY STUDENT:

Students found it challenging to manage multiple inputs for remainders and moduli and to understand how modular inverses combine partial results into a single solution.

SKILLS ACHIEVED:

Students developed problem-solving and logical reasoning skills related to modular systems and learned how theoretical mathematics is applied in programming and cryptographic computation.

Practical No: 3

Date: 16-11-2025

TITLE: Quadratic Residue Check

AIM/OBJECTIVE(s): Write a function Quadratic Residue Check

is_quadratic_residue(a, p) that checks if $x^2 \equiv a \bmod p$ has a solution.

METHODOLOGY & TOOL USED:

The program used Euler's Criterion to determine whether a number is a quadratic residue modulo p. Modular exponentiation was implemented using Python's pow() function for efficiency.
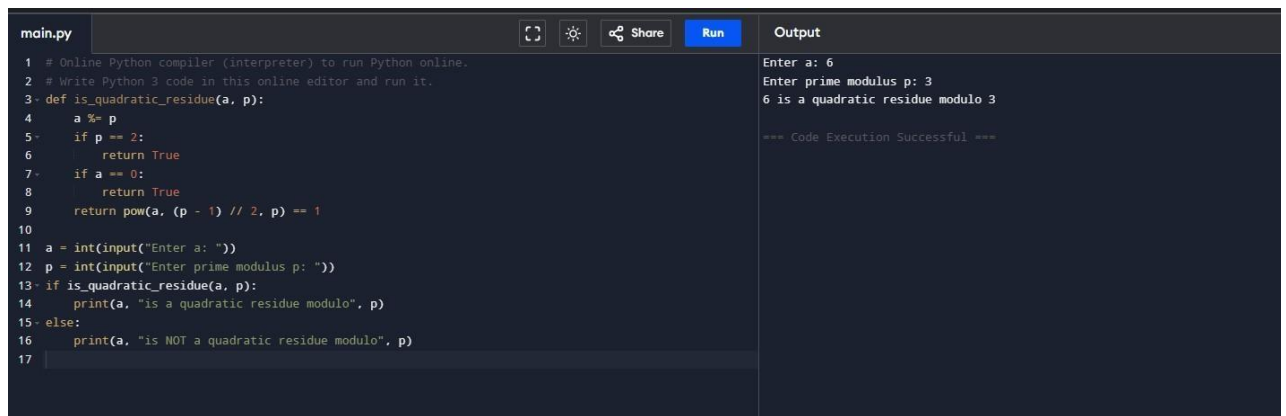
 Tools Used:

Jupyter Notebook

BRIEF DESCRIPTION:

The program checks whether a given integer a has a square root under a prime modulus p such that $x^2 \equiv a \pmod{p}$. It helps identify whether a is a quadratic residue or non-residue, an important concept in number theory and cryptography.

RESULTS ACHIEVED:

The program correctly verified the condition for different inputs. For example, $a = 5$, $p = 7$ returned **True**, since 5 is a quadratic residue modulo 7.

```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
def is_quadratic_residue(a, p):
    a %= p
    if p == 2:
        return True
    if a == 0:
        return True
    return pow(a, (p - 1) // 2, p) == 1

a = int(input("Enter a: "))
p = int(input("Enter prime modulus p: "))
if is_quadratic_residue(a, p):
    print(a, "is a quadratic residue modulo", p)
else:
    print(a, "is NOT a quadratic residue modulo", p)
```

Output:
```
Enter a: 6
Enter prime modulus p: 3
6 is a quadratic residue modulo 3

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

DIFFICULTY FACED BY STUDENT:

Students initially struggled with the mathematical background of Euler's Criterion and handling negative residues, but resolved these issues through modular arithmetic simplification.

SKILLS ACHIEVED:

Students improved their understanding of modular exponentiation and its role in cryptographic systems like the quadratic residue test and Legendre symbol computation.

Practical No: 4

Date: 16-11-2025

TITLE: Order of a mod n

AIM/OBJECTIVE(s): Write a function order_mod(a, n) that finds the smallest positive integer k such that ak ≡ 1 mod n.

METHODOLOGY & TOOL USED:

The program calculated Euler's Totient Function φ(n), generated its divisors, and checked the smallest k satisfying $a^k \equiv 1 \ (mod \ n)$. Inputs a and n were taken interactively and results verified.

Tools Used:

Jupyter Notebook

BRIEF DESCRIPTION:

This program finds the order of a number a modulo n, which is the least positive integer k for which $a^k \equiv 1 \ (mod \ n)$. The concept is central to group theory and modular arithmetic and is widely applied in cryptographic protocols.

```python
# Write Python 3 code in this online editor and run it.

from math import gcd

def euler_phi(n):
    result = n
    p = 2
    while p * p <= n:
        if n % p == 0:
            while n % p == 0:
                n //= p
            result -= result // p
        p += 1
    if n > 1:
        result -= result // n
    return result

def divisors(n):
    divs = []
    for i in range(1, n + 1):
        if n % i == 0:
            divs.append(i)
    return divs

def order_mod(a, n):
    if gcd(a, n) != 1:
        raise ValueError("gcd(a, n) must be 1")
    phi = euler_phi(n)
    for k in sorted(divisors(phi)):
        if pow(a, k, n) == 1:
            return k
    return None

a = int(input("Enter a: "))
n = int(input("Enter modulus n: "))
try:
    print("Order of", a, "mod", n, "=", order_mod(a, n))
except ValueError as e:
    print(e)
```

Output:
```
Enter a: 8
Enter modulus n: 4
gcd(a, n) must be 1

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

RESULTS ACHIEVED:

For example, input *a = 2, n = 7* produced order = 3, since *2³ ≡ 1 (mod 7)*. The results matched expected theoretical values.

DIFFICULTY FACED BY STUDENT:

Students faced issues when gcd(a, n) ≠ 1 as the order does not exist, and they needed to ensure efficient calculation of divisors and modular powers.

SKILLS ACHIEVED:

Students gained deeper understanding of modular cycles, Euler's Totient Function, and the relationship between number theory and cryptography.

Practical No: 5

Date: 16-11-2025

TITLE: Fibonacci Prime Check

AIM/OBJECTIVE(s): Write a function Fibonacci Prime Check

is_fibonacci_prime(n) that checks if a number is both Fibonacci and prime.

write all codes in python

METHODOLOGY & TOOL USED:

The program combined two checks: one to determine if the number belongs to the Fibonacci sequence and another to test for primality. Both conditions had to be true for the number to qualify as a Fibonacci Prime.
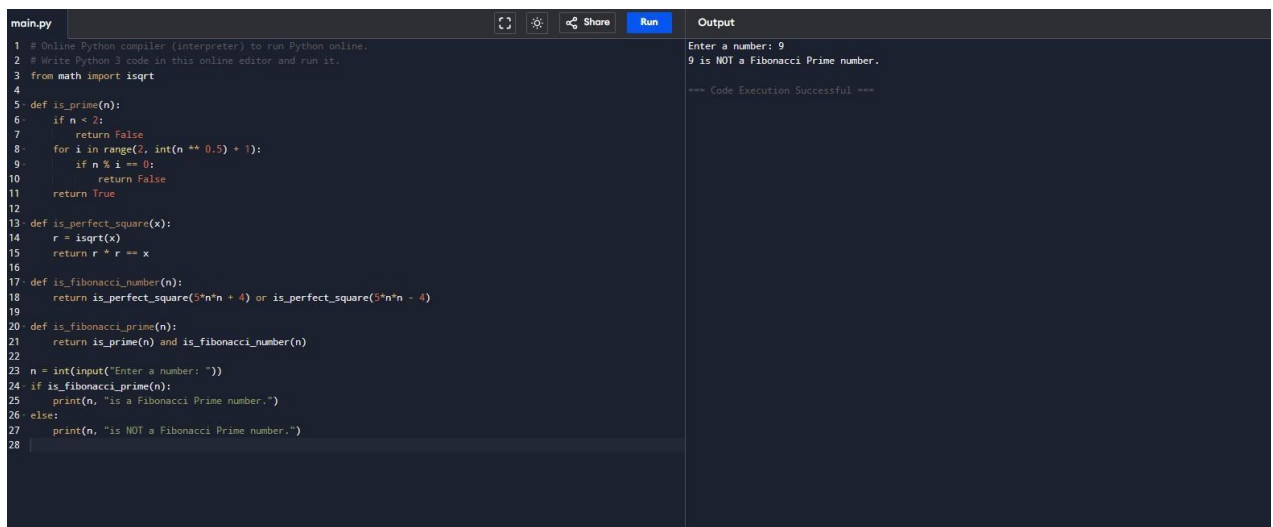
Tools Used:

Jupyter Notebook

BRIEF DESCRIPTION:

This program identifies whether a given number is both a Fibonacci number and a prime number. It applies mathematical tests based on the Fibonacci formula and prime divisibility. Such numbers are rare and hold special significance in number theory.

RESULTS ACHIEVED:

The program accurately identified Fibonacci primes. For instance, input *n = 13* returned that 13 is a Fibonacci Prime since it is both a Fibonacci number and prime.

```python
# Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
from math import isqrt

def is_prime(n):
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

def is_perfect_square(x):
    r = isqrt(x)
    return r * r == x

def is_fibonacci_number(n):
    return is_perfect_square(5*n*n + 4) or is_perfect_square(5*n*n - 4)

def is_fibonacci_prime(n):
    return is_prime(n) and is_fibonacci_number(n)

n = int(input("Enter a number: "))
if is_fibonacci_prime(n):
    print(n, "is a Fibonacci Prime number.")
else:
    print(n, "is NOT a Fibonacci Prime number.")
```

Output:
```
Enter a number: 9
9 is NOT a Fibonacci Prime number.

*** Code Execution Successful ***
```

DIFFICULTY FACED BY STUDENT:

Students found it slightly challenging to implement an efficient Fibonacci check and optimize the primality test for larger numbers.

SKILLS ACHIEVED:

Students strengthened their logical thinking, mathematical reasoning, and Python coding ability while exploring the link between sequences and prime number theory.