



Lab Manual

Practical and Skills Development

CERTIFICATE

THE ASSIGNMENT ENTERED IN THIS REPORT HAVE BEEN SATISFACTORILY
PERFORMED BY

Registration No : 25BAI10749
Name of Student : Aniruddha Majumdar

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	14-11-2025	
2	Chinese Remainder Theorem Solver	14-11-2025	
3	Quadratic Residue Check	14-11-2025	
4	Order of a mod n	14-11-2025	

5	Fibonacci Prime Check	14-11-2025	
----------	------------------------------	-------------------	--

6			
7			
8			
9			
10			
11			

12			
13			
14			
15			

Practical No: 1

Date: 14-11-2025

TITLE: Modular Multiplicative Inverse

AIM/OBJECTIVE(s) : Write a function Modular Multiplicative Inverse `mod_inverse(a, m)` that finds the number x such that $(a * x) \equiv 1 \pmod{m}$.

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 $(a \times x) \equiv 1 \pmod{m}$. 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.

main.py	Run	Output
<pre> 1 # Online Python compiler (interpreter) to run Python online. 2 # Write Python 3 code in this online editor and run it. 3- def egcd(a, b): 4 """Extended Euclidean Algorithm""" 5- if b == 0: 6- return (a, 1, 0) 7- g, x1, y1 = egcd(b, a % b) 8- return (g, y1, x1 - (a // b) * y1) 9 10- def mod_inverse(a, m): 11 """Finds x such that (a * x) ≡ 1 (mod m)""" 12 a = a % m 13 g, x, _ = egcd(a, m) 14 if g != 1: 15 raise ValueError("Inverse does not exist") 16 return x % m 17 18 # ---- Input Section ---- 19 a = int(input("Enter a: ")) 20 m = int(input("Enter modulus m: ")) 21 try: 22 print("Modular Inverse =", mod_inverse(a, m)) 23 except ValueError as e: 24 print(e) 25 </pre>	Run	<pre> Enter a: 8 Enter modulus m: 9 Modular Inverse = 8 ==== Code Execution Successful ==== </pre>



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: 14-11-2025

TITLE: Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences $x \equiv r_i \pmod{m_i}$.

AIM/OBJECTIVE(s): Write a function chinese Remainder Theorem Solver crt(remainders, moduli) that solves a system of congruences $x \equiv r_i \pmod{m_i}$.



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 $x \equiv r_i \pmod{m_i}$ 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 $x \equiv 2 \pmod{3}$, $x \equiv 3 \pmod{5}$, $x \equiv 2 \pmod{7}$, the result was $x = 23 \pmod{105}$.

<pre> main.py 1 # Online Python compiler (interpreter) to run Python online. 2 # Write Python 3 code in this online editor and run it. 3 def chinese_remainder_theorem(remainders, moduli): 4 M = 1 5 for m in moduli: 6 M *= m 7 x = 0 8 for r, m in zip(remainders, moduli): 9 Mi = M // m 10 inv = pow(Mi, -1, m) 11 x += r * Mi * inv 12 return x % M 13 14 15 print("Chinese Remainder Theorem Solver") 16 n = int(input("Enter the number of congruences: ")) 17 18 remainders = [] 19 moduli = [] 20 21 for i in range(1, n + 1): 22 print(f"\nCongruence {i}:") 23 r = int(input(" Enter remainder r: ")) 24 m = int(input(" Enter modulus m: ")) 25 remainders.append(r) 26 moduli.append(m) 27 28 try: 29 result = chinese_remainder_theorem(remainders, moduli) 30 print("\nThe smallest x that satisfies all congruences is:", result) 31 except Exception as e: 32 print("\nError:", e) 33 </pre>	<p>Output</p> <pre> Chinese Remainder Theorem Solver Enter the number of congruences: 2 Congruence 1: Enter remainder r: 5 Enter modulus m: 3 Congruence 2: Enter remainder r: 7 Enter modulus m: 7 The smallest x that satisfies all congruences is: 14 *** Code Execution Successful *** </pre>
---	---

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: 14-11-2025



TITLE: Quadratic Residue Check

AIM/OBJECTIVE(s): Write a function Quadratic Residue Check
is_quadratic_residue(a, p) that checks if $x^2 \equiv a \pmod{p}$ 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

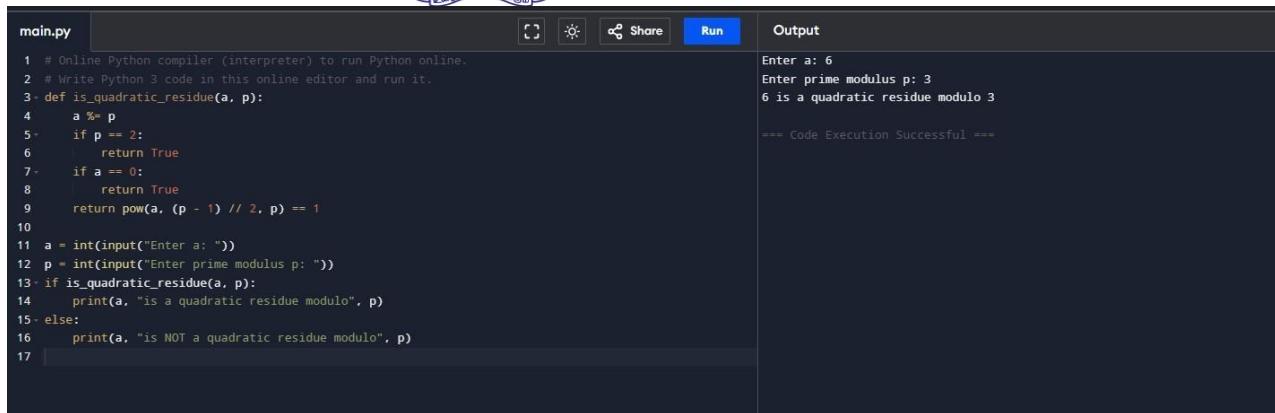
BRIEF DESCRIPTION:

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.



```
main.py
1 # Online Python compiler (interpreter) to run Python online.
2 # Write Python 3 code in this online editor and run it.
3 def is_quadratic_residue(a, p):
4     a %= p
5     if p == 2:
6         return True
7     if a == 0:
8         return True
9     return pow(a, (p - 1) // 2, p) == 1
10
11 a = int(input("Enter a: "))
12 p = int(input("Enter prime modulus p: "))
13 if is_quadratic_residue(a, p):
14     print(a, "is a quadratic residue modulo", p)
15 else:
16     print(a, "is NOT a quadratic residue modulo", p)
17 
```

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: 14-11-2025

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 $a^k \equiv 1 \pmod{n}$.

METHODOLOGY & TOOL USED:

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

Tools Used:

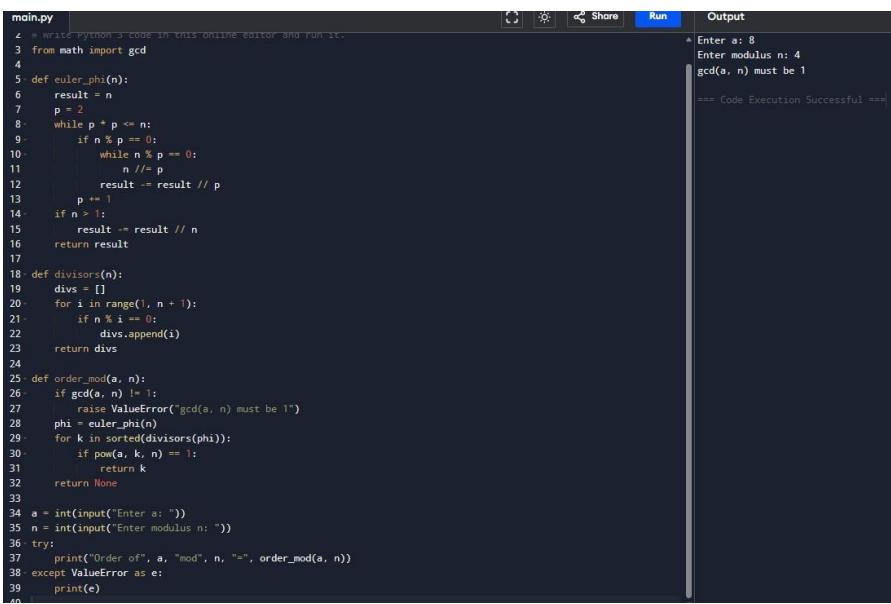
Jupyter Notebook

BRIEF DESCRIPTION:

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

RESULTS ACHIEVED:

For example, input $a = 2, n = 7$ produced order = 3, since $2^3 \equiv 1 \pmod{7}$. The results matched expected theoretical values.



```

main.py
1 # Write Python 3 code in this online editor and run it.
2
3 from math import gcd
4
5 def euler_phi(n):
6     result = n
7     p = 2
8     while p * p <= n:
9         if n % p == 0:
10             while n % p == 0:
11                 n /= p
12             result *= result // p
13         p += 1
14     if n > 1:
15         result *= result // n
16     return result
17
18 def divisors(n):
19     divs = []
20     for i in range(1, n + 1):
21         if n % i == 0:
22             divs.append(i)
23     return divs
24
25 def order_mod(a, n):
26     if gcd(a, n) != 1:
27         raise ValueError("gcd(a, n) must be 1")
28     phi = euler_phi(n)
29     for k in sorted(divisors(phi)):
30         if pow(a, k, n) == 1:
31             return k
32     return None
33
34 a = int(input("Enter a: "))
35 n = int(input("Enter modulus n: "))
36 try:
37     print("Order of", a, "mod", n, "=", order_mod(a, n))
38 except ValueError as e:
39     print(e)
40

```

DIFFICULTY FACED BY STUDENT:

Students faced issues when $\text{gcd}(a, n) \neq 1$ 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: 14-11-2025

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:

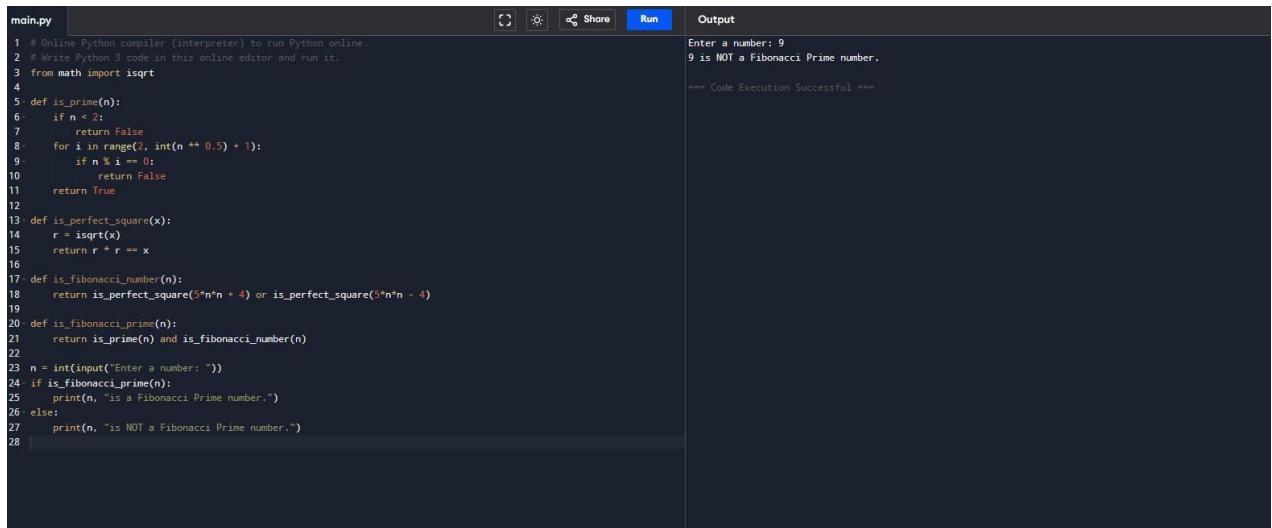
Jupyter Notebook

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.



```

main.py
1 # Online Python compiler (interpreter) to run Python online.
2 # Write Python 3 code in this online editor and run it.
3 from math import isqrt
4
5 def is_prime(n):
6     if n < 2:
7         return False
8     for i in range(2, int(n ** 0.5) + 1):
9         if n % i == 0:
10             return False
11     return True
12
13 def is_perfect_square(x):
14     r = isqrt(x)
15     return r * r == x
16
17 def is_fibonacci_number(n):
18     return is_perfect_square(5*n*n + 4) or is_perfect_square(5*n*n - 4)
19
20 def is_fibonacci_prime(n):
21     return is_prime(n) and is_fibonacci_number(n)
22
23 n = int(input("Enter a number: "))
24 if is_fibonacci_prime(n):
25     print(n, "is a Fibonacci Prime number.")
26 else:
27     print(n, "is NOT a Fibonacci Prime number.")
28

```

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.