

# AI ASSISTED CODING

SUMANTH POLAM

2303A51121

BATCH – 03

30 – 01 – 2026

## ASSIGNMENT – 5.5

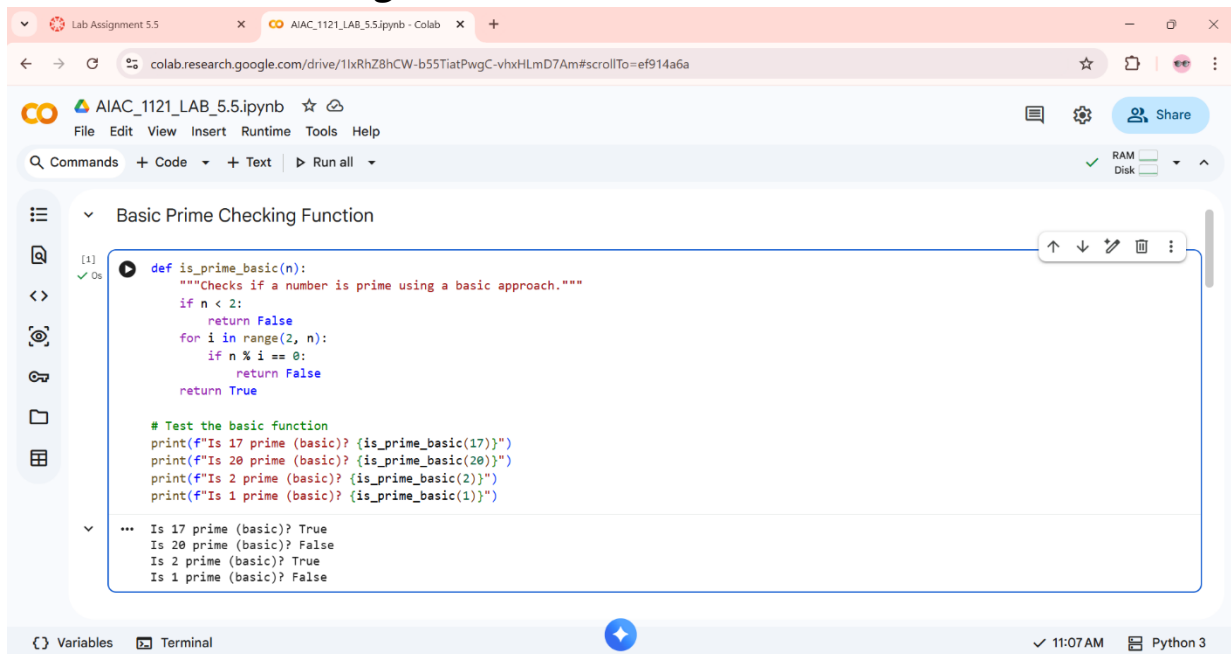
**Lab 5:** Ethical Foundations – Responsible AI Coding Practices.

**TASK - 01 :** (Transparency in Algorithm Optimization)

**Prompt :** Generate Python code for two prime-checking methods and explain how the optimized version improves performance.

**Code:**

### 1. Basic Prime Checking Function

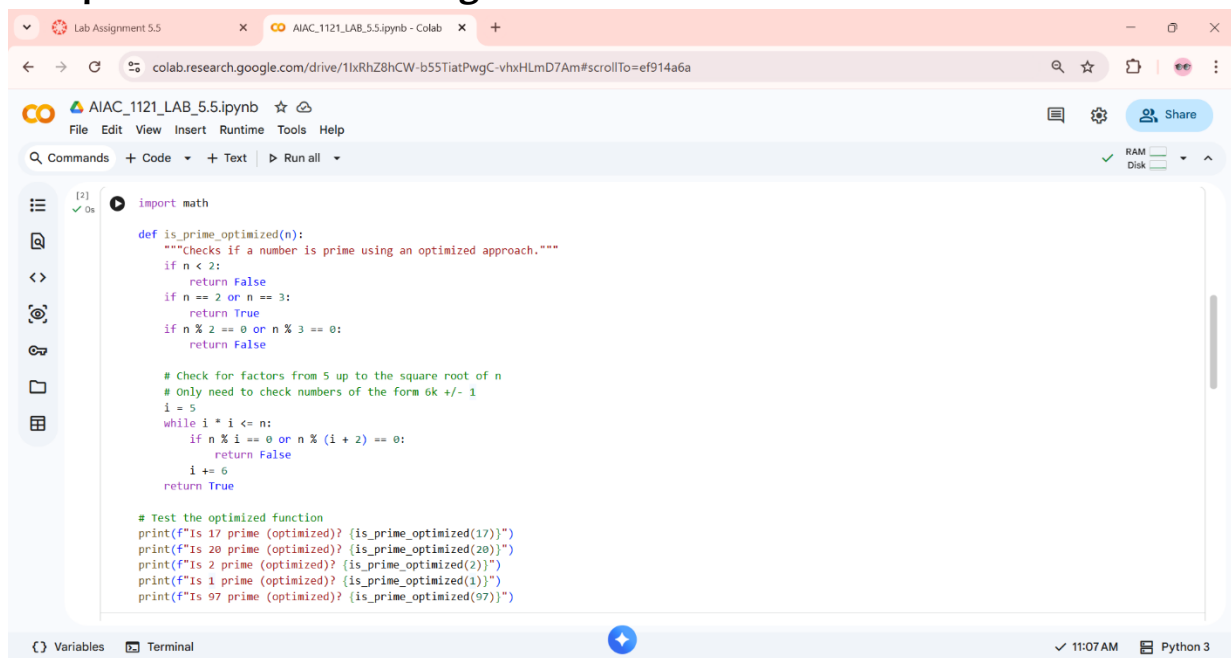


```
def is_prime_basic(n):
    """Checks if a number is prime using a basic approach."""
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

# Test the basic function
print(f"Is 17 prime (basic)? {is_prime_basic(17)}")
print(f"Is 20 prime (basic)? {is_prime_basic(20)}")
print(f"Is 2 prime (basic)? {is_prime_basic(2)}")
print(f"Is 1 prime (basic)? {is_prime_basic(1)}")
```

\*\*\* Is 17 prime (basic)? True  
Is 20 prime (basic)? False  
Is 2 prime (basic)? True  
Is 1 prime (basic)? False

## 2. Optimized Prime Checking Function



The screenshot shows a Google Colab notebook with the following Python code:

```
import math

def is_prime_optimized(n):
    """Checks if a number is prime using an optimized approach."""
    if n < 2:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False

    # Check for factors from 5 up to the square root of n
    # Only need to check numbers of the form 6k +/- 1
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True

# Test the optimized function
print(f"Is 17 prime (optimized)? {is_prime_optimized(17)}")
print(f"Is 20 prime (optimized)? {is_prime_optimized(20)}")
print(f"Is 2 prime (optimized)? {is_prime_optimized(2)}")
print(f"Is 1 prime (optimized)? {is_prime_optimized(1)}")
print(f"Is 97 prime (optimized)? {is_prime_optimized(97)}")
```

### Transparent Explanation:

- ✓ Naive Method Time Complexity:  $O(n)$   
→ Checks all numbers from 2 to  $n-1$ .
- ✓ Optimized Method Time Complexity:  $O(\sqrt{n})$   
→ Only checks up to square root of  $n$ .

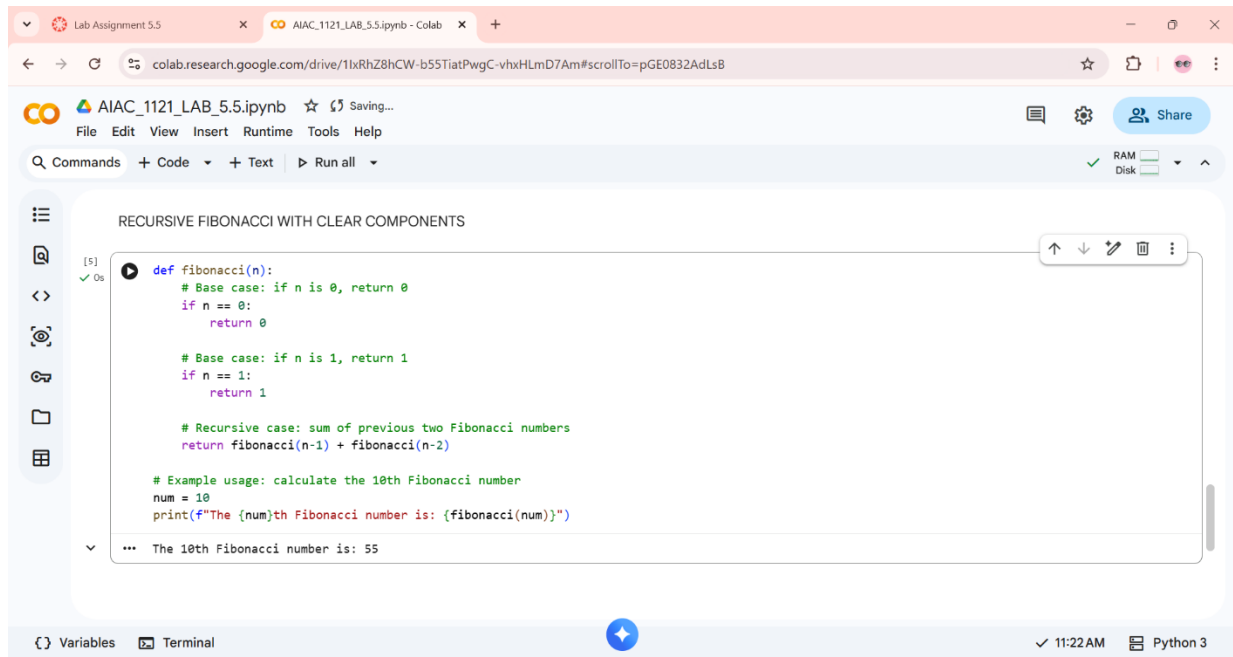
### Comparison :

<u>Method</u>	<u>Time Complexity</u>	<u>Performance</u>
Naive	$O(n)$	Slower
Optimized	$O(\sqrt{n})$	Faster

**Task – 02 :** Transparency in Recursive Algorithms.

**Prompt :** Give me the Recursive Fibonacci code with clear comments.

## Code:



Lab Assignment 5.5

AIAC\_1121\_LAB\_5.5.ipynb - Colab

colab.research.google.com/drive/1xRhZ8hCW-b55TiatPwgC-vhxHLMd7Am#scrollTo=pGE0832AdLsB

AIAC\_1121\_LAB\_5.5.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RECURSIVE FIBONACCI WITH CLEAR COMPONENTS

```
[5] def fibonacci(n):  
    # Base case: if n is 0, return 0  
    if n == 0:  
        return 0  
  
    # Base case: if n is 1, return 1  
    if n == 1:  
        return 1  
  
    # Recursive case: sum of previous two Fibonacci numbers  
    return fibonacci(n-1) + fibonacci(n-2)  
  
# Example usage: calculate the 10th Fibonacci number  
num = 10  
print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
```

... The 10th Fibonacci number is: 55

Variables Terminal

11:22 AM Python 3

## Explanation:

- Base Cases:
  - $\text{fibonacci}(0) \rightarrow 0$
  - $\text{fibonacci}(1) \rightarrow 1$
- Recursive Call:
  - $\text{fibonacci}(n) = \text{fibonacci}(n-1) + \text{fibonacci}(n-2)$

## Task – 03 : Transparency in Error Handling.

**Prompt :** Generate code with proper error handling and clear explanations for each exception.

## Code:

The image shows two screenshots of a Google Colab notebook titled 'AIAC\_1121\_LAB\_5.5.ipynb'. The first screenshot displays the definition of a `fibonacci(n)` function. It includes input validation for non-integers and negative numbers, base cases for `n=0` and `n=1`, and a recursive case. An example usage with error handling is shown at the bottom, testing with `num = 10`. The second screenshot shows the execution of the function with various inputs and corresponding error messages. It tests with `num = 10` (successful), `num = -5` (ValueError), `num = 5.5` (ValueError), and `num = 'abc'` (TypeError).

```
def fibonacci(n):
    # Input validation
    if not isinstance(n, int):
        raise TypeError("Input must be an integer.")
    if n < 0:
        raise ValueError("Input cannot be a negative number.")

    # Base case: if n is 0, return 0
    if n == 0:
        return 0

    # Base case: if n is 1, return 1
    if n == 1:
        return 1

    # Recursive case: sum of previous two Fibonacci numbers
    return fibonacci(n-1) + fibonacci(n-2)

# Example usage with error handling:
# Test with valid input
try:
    num = 10
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
except (TypeError, ValueError) as e:
    print(f"Error for input {num}: {e}")

# Test with negative input
try:
    num = -5
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
except (TypeError, ValueError) as e:
    print(f"Error for input {num}: {e}")

# Test with non-integer input
try:
    num = 5.5
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
except (TypeError, ValueError) as e:
    print(f"Error for input {num}: {e}")

# Test with string input
try:
    num = "abc"
    print(f"The {num}th Fibonacci number is: {fibonacci(num)}")
except (TypeError, ValueError) as e:
    print(f"Error for input '{num}': {e}")
```

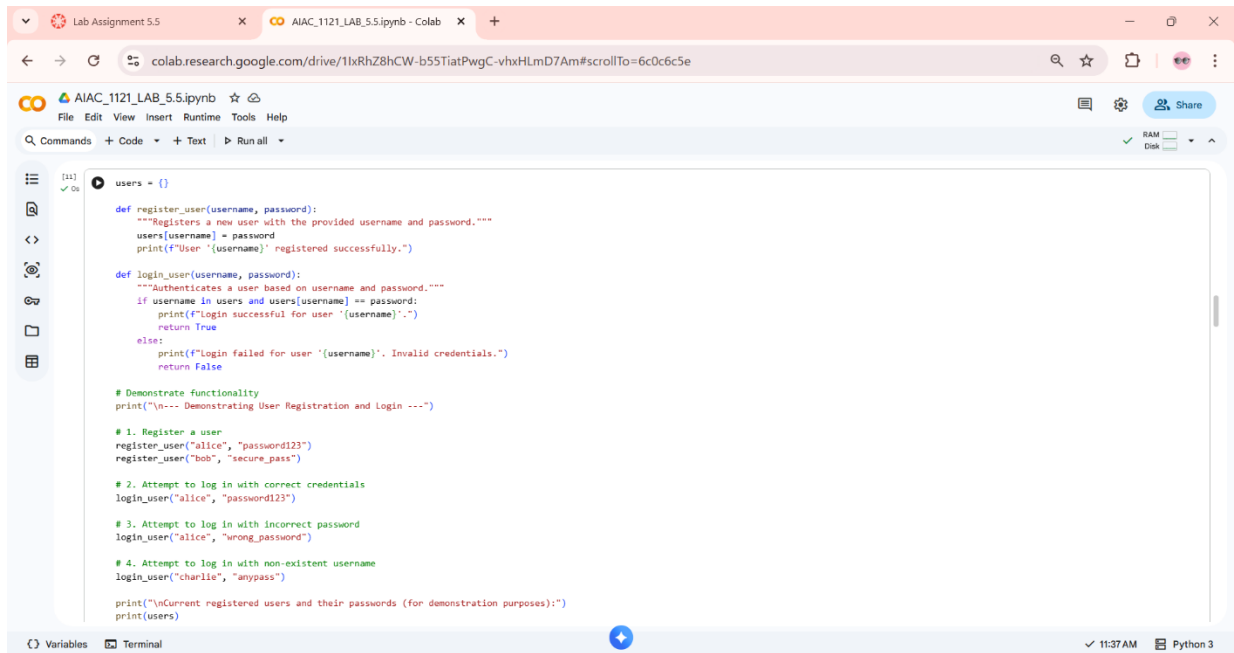
## Explaining the Errors:

Exception	Meaning
<code>FileNotFoundError</code>	File does not exist
<code>PermissionError</code>	No permission to read file
Exception	Any other unknown error

## Task – 04 : Security in User Authentication.

### Code:

#### Insecure Version:



The screenshot shows a Google Colab notebook titled "AIAC\_1121\_LAB\_5.5.ipynb". The code defines a simple user authentication system with a dictionary for users. It includes functions for registering a user and logging in. The login function checks if the username exists and if the password matches. The code demonstrates functionality by registering two users and attempting to log in with correct and incorrect credentials.

```
[11]: users = {}

def register_user(username, password):
    """Registers a new user with the provided username and password."""
    users[username] = password
    print(f"User '{username}' registered successfully.")

def login_user(username, password):
    """Authenticates a user based on username and password."""
    if username in users and users[username] == password:
        print(f"Login successful for user '{username}'.")
        return True
    else:
        print(f"Login failed for user '{username}'. Invalid credentials.")
        return False

# Demonstrate functionality
print("\n--- Demonstrating User Registration and Login ---")

# 1. Register a user
register_user("alice", "password123")
register_user("bob", "secure_pass")

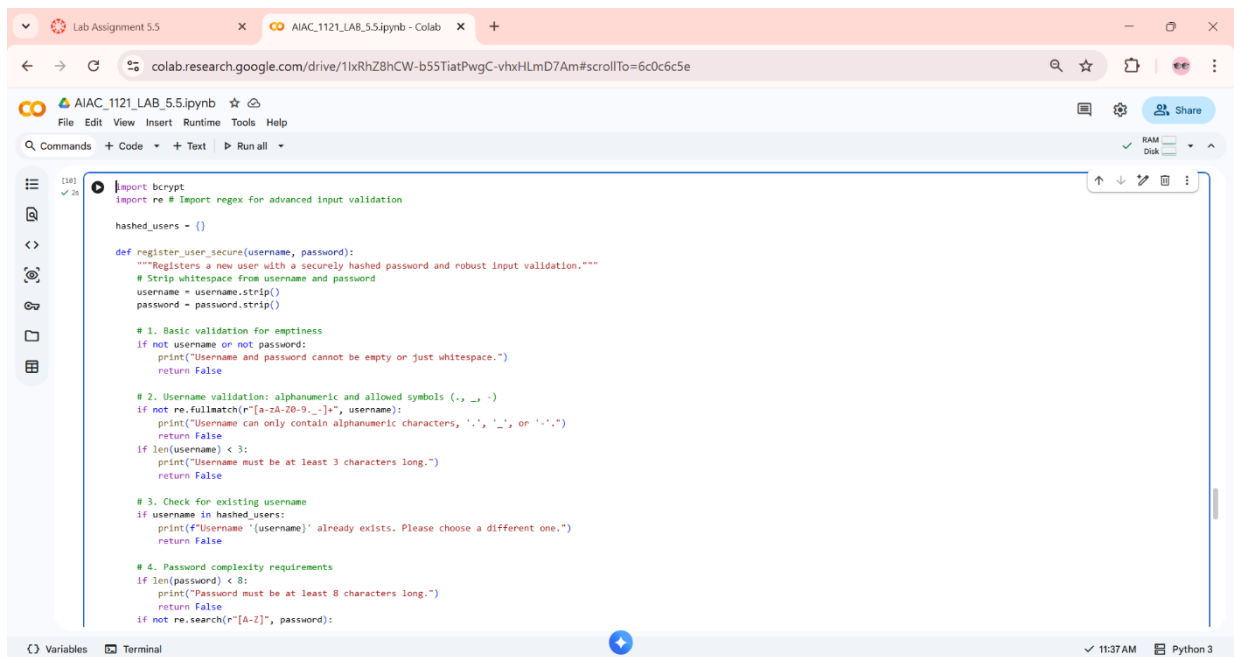
# 2. Attempt to log in with correct credentials
login_user("alice", "password123")

# 3. Attempt to log in with incorrect password
login_user("alice", "wrong_password")

# 4. Attempt to log in with non-existent username
login_user("charlie", "anypass")

print("\nCurrent registered users and their passwords (for demonstration purposes):")
print(users)
```

#### Secure Version:



The screenshot shows a Google Colab notebook titled "AIAC\_1121\_LAB\_5.5.ipynb". The code imports the bcrypt module and the re module for regex. It defines a secure user authentication system with a dictionary for hashed users. The register\_user\_secure function includes input validation for username and password, checks for existing usernames, and uses bcrypt to hash passwords. The login function checks if the username exists and if the password matches the hashed password.

```
[18]: import bcrypt
import re # Import regex for advanced input validation

hashed_users = {}

def register_user_secure(username, password):
    """Registers a new user with a securely hashed password and robust input validation."""
    # Strip whitespace from username and password
    username = username.strip()
    password = password.strip()

    # 1. Basic validation for emptiness
    if not username or not password:
        print("Username and password cannot be empty or just whitespace.")
        return False

    # 2. Username validation: alphanumeric and allowed symbols (.,_,-)
    if not re.fullmatch(r"[a-zA-Z0-9-_.]*", username):
        print("Username can only contain alphanumeric characters, '.', '_', or '-'")
        return False

    if len(username) < 3:
        print("Username must be at least 3 characters long.")
        return False

    # 3. Check for existing username
    if username in hashed_users:
        print(f"Username '{username}' already exists. Please choose a different one.")
        return False

    # 4. Password complexity requirements
    if len(password) < 8:
        print("Password must be at least 8 characters long.")
        return False

    if not re.search(r"[A-Z]", password):
        print("Password must contain at least one uppercase letter.")
        return False

    # Hash the password
    hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

    hashed_users[username] = hashed_password
    print(f"User '{username}' registered successfully with secure hashing.")

# Demonstrate secure functionality
print("\n--- Demonstrating Secure User Registration and Login ---")

# 1. Register a user
register_user_secure("alice", "password123")
register_user_secure("bob", "secure_pass")

# 2. Attempt to log in with correct credentials
login_user_secure("alice", "password123")

# 3. Attempt to log in with incorrect password
login_user_secure("alice", "wrong_password")

# 4. Attempt to log in with non-existent username
login_user_secure("charlie", "anypass")

print("\nCurrent registered users and their hashed passwords (for demonstration purposes):")
print(hashed_users)
```

The image displays two screenshots of a Google Colab notebook titled 'AIAC\_1121\_LAB\_5.5.ipynb'. The first screenshot shows the initial code for user registration and login. The second screenshot shows the code after execution, with output messages displayed at the bottom.

```
def login_user_secure(username, password):
    """Authenticates a user against their securely hashed password with input stripping."""
    # Strip whitespace from username and password
    username = username.strip()
    password = password.strip()

    if username not in hashed_users:
        print("Login failed: Invalid credentials.") # Generic message for security
        return False

    # Check the provided password against the stored hash
    if bcrypt.checkpw(password.encode('utf-8'), hashed_users[username]):
        print(f"Login successful for user '{username}'.")
        return True
    else:
        print("Login failed: Invalid credentials.") # Generic message for security
        return False

# Demonstrate functionality with enhanced secure system
print("\n--- Demonstrating Enhanced Secure User Registration and Login ---")

# 1. Register users with new validations
register_user_secure("jane_doe", "StrongPass1!")
register_user_secure("user with space", "ValidPass2@") # Invalid username
register_user_secure("anotherUser", "ValidPass3#") # Invalid username
register_user_secure("bob", "weak") # Password too short
register_user_secure("bob", "onlylowercase") # Missing uppercase, digit, special
register_user_secure("david", "SecurePass4") # Missing special character
register_user_secure("emily", "emily123!") # Valid password, but username exists
register_user_secure("emily", "EmilyP4ss") # Valid registration

# 2. Demonstrate stripping whitespace
register_user_secure(" padded_user ", " PaddedPass5$ ") # Should register 'padded_user'
login_user_secure("padded_user", "PaddedPass5$")
login_user_secure(" padded_user ", "PaddedPass5$") # Login with padded username
login_user_secure("padded_user", " PaddedPass5$ ") # Login with padded password

# 3. Attempt to log in with correct credentials
login_user_secure("jane_doe", "StrongPass1!")

# 4. Attempt to log in with incorrect password
login_user_secure("jane_doe", "wrong_password")

# 5. Attempt to log in with non-existent username
login_user_secure("frank", "anypass")

print("\nCurrent registered users (hashed passwords stored, not displayed for security):")
print(f"Users registered: {list(hashed_users.keys())}")

--- Demonstrating Enhanced Secure User Registration and Login ---
User 'jane_doe' registered securely.
Username can only contain alphanumeric characters, '.', '-', '_' or '@'.
Username can only contain alphanumeric characters, '.', '-', '_' or '@'.
Password must be at least 8 characters long.
Password must contain at least one uppercase letter.
```

The second screenshot shows the output of the code execution, including the registration of 'jane\_doe' and the list of registered users.

## Explanation :

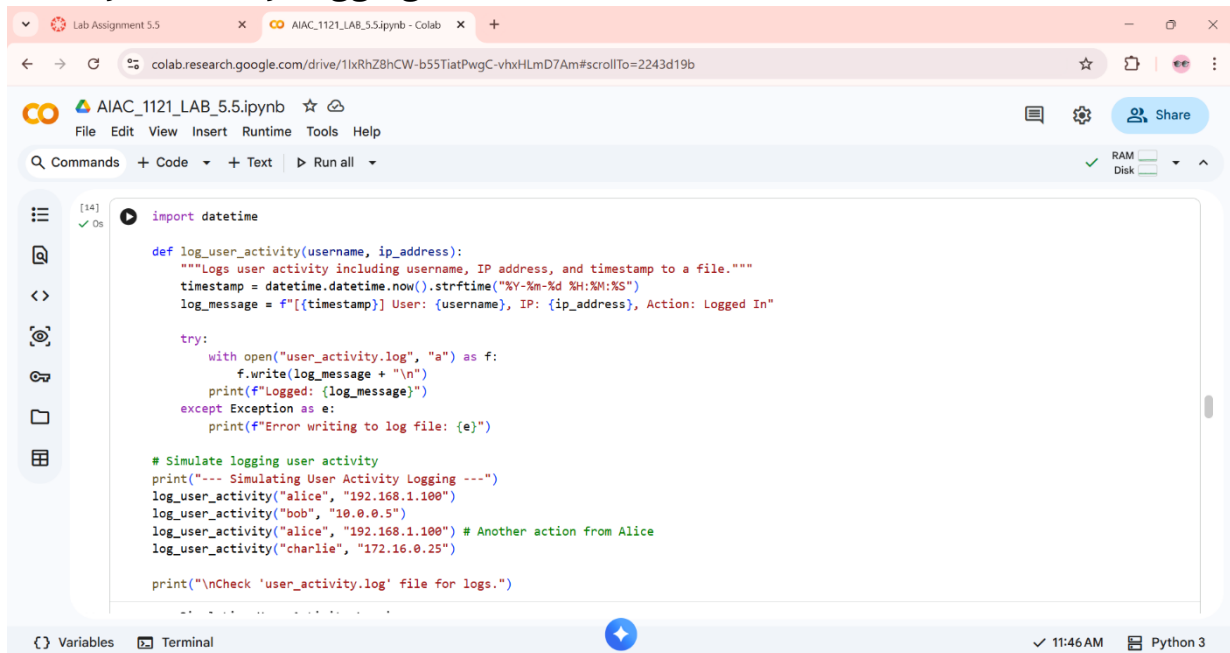
- ✓ Always hash passwords
- ✓ Never store plain-text passwords
- ✓ Validate user input
- ✓ Use strong hashing algorithms

## Task – 05 : Privacy in Data Logging.

**Prompt – 01 :** Create a basic Python script that simulates logging user activity, including username, IP address, and timestamp, to a file or console.

## Code:

### Privacy and Risky Logging:



```
[14] ✓ 0s
import datetime

def log_user_activity(username, ip_address):
    """Logs user activity including username, IP address, and timestamp to a file."""
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_message = f"[{timestamp}] User: {username}, IP: {ip_address}, Action: Logged In"

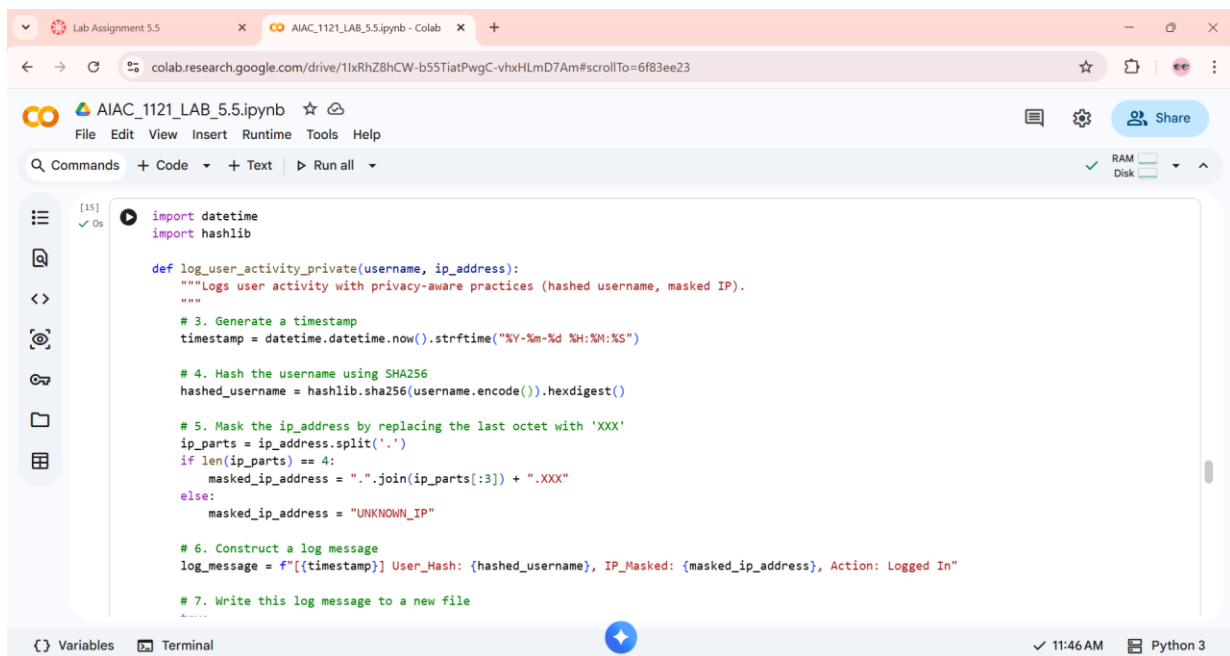
    try:
        with open("user_activity.log", "a") as f:
            f.write(log_message + "\n")
        print(f"Logged: {log_message}")
    except Exception as e:
        print(f"Error writing to log file: {e}")

# Simulate logging user activity
print("--- Simulating User Activity Logging ---")
log_user_activity("alice", "192.168.1.100")
log_user_activity("bob", "10.0.0.5")
log_user_activity("alice", "192.168.1.100") # Another action from Alice
log_user_activity("charlie", "172.16.0.25")

print("\nCheck 'user_activity.log' file for logs.")
```

**Prompt – 02 :** Examine the initial logging script to identify specific privacy risks associated with logging sensitive data like usernames and IP addresses directly. Detail potential negative impacts.

## Code:



```
[15] ✓ 0s
import datetime
import hashlib

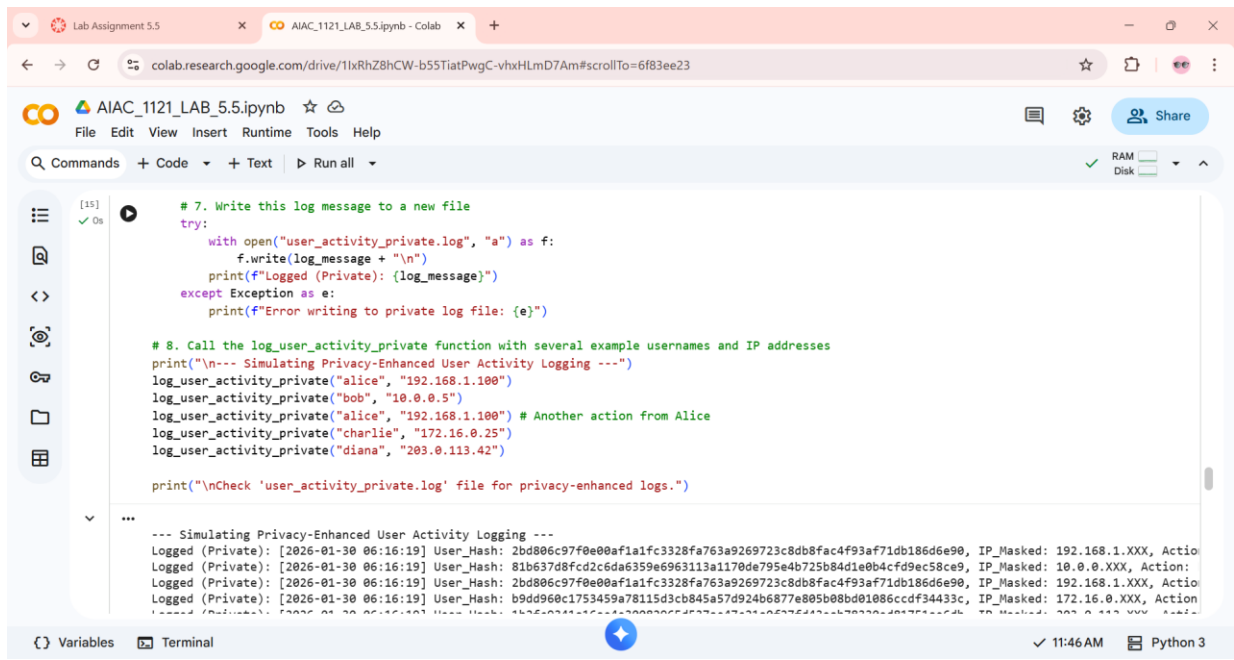
def log_user_activity_private(username, ip_address):
    """Logs user activity with privacy-aware practices (hashed username, masked IP)."""
    # 3. Generate a timestamp
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

    # 4. Hash the username using SHA256
    hashed_username = hashlib.sha256(username.encode()).hexdigest()

    # 5. Mask the ip_address by replacing the last octet with 'XXX'
    ip_parts = ip_address.split('.')
    if len(ip_parts) == 4:
        masked_ip_address = ".".join(ip_parts[:3]) + ".XXX"
    else:
        masked_ip_address = "UNKNOWN_IP"

    # 6. Construct a log message
    log_message = f"[{timestamp}] User_Hash: {hashed_username}, IP_Masked: {masked_ip_address}, Action: Logged In"

    # 7. Write this log message to a new file
```



**Explanation :**

- ✓ Mask or anonymize sensitive data
- ✓ Log only what is necessary
- ✓ Avoid storing personal identifiers
- ✓ Protect log files from unauthorized access

## THANK YOU!!