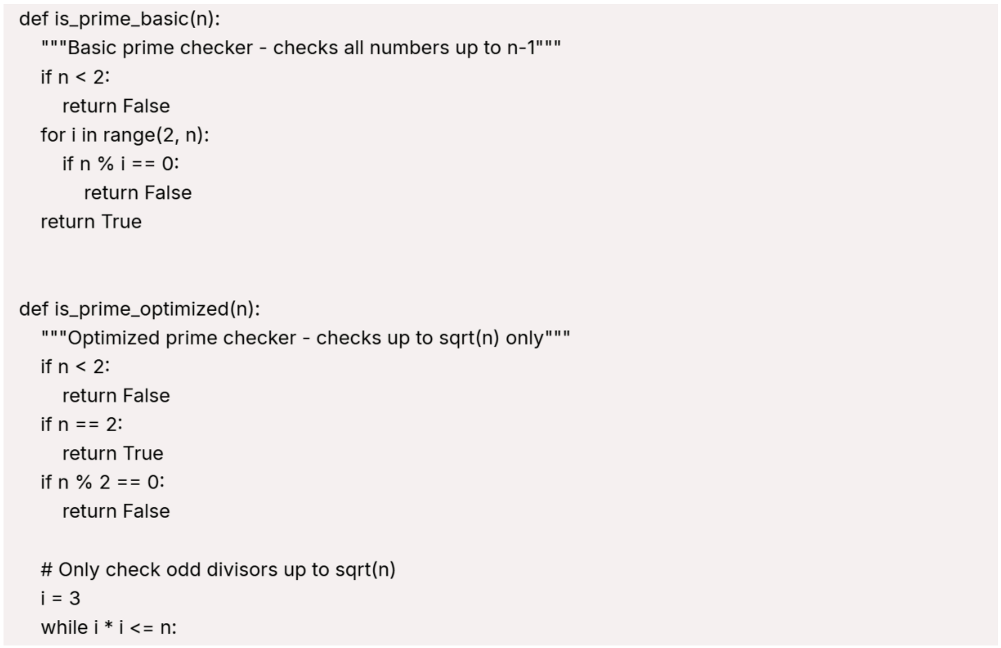
AI ASSISTED CODING

ASSIGNMENT-5.5

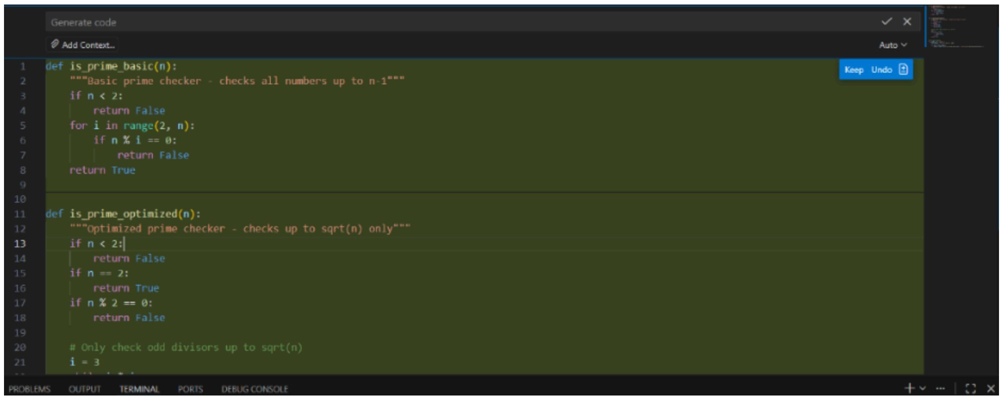
2303A51918

BT:30

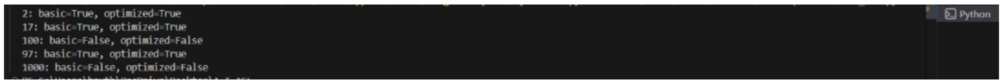
Task Description #1 (Transparency in Algorithm Optimization)
Task: Use AI to generate two solutions for checking prime numbers:
• Naive approach(basic)
• Optimized approach
Prompt:
“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”
Expected Output:
• Code for both methods.
• Transparent explanation of time complexity.
• Comparison highlighting efficiency improvements.



```
def is_prime_basic(n):
    """Basic prime checker - checks all numbers up to n-1"""
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True


def is_prime_optimized(n):
    """Optimized prime checker - checks up to sqrt(n) only"""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Only check odd divisors up to sqrt(n)
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2
    return True


# Test both methods
if __name__ == "__main__":
    test_numbers = [2, 17, 100, 97, 1000]

    for num in test_numbers:
        print(f"{num}: basic={is_prime_basic(num)}, optimized={is_prime_optimized(num)}")
```

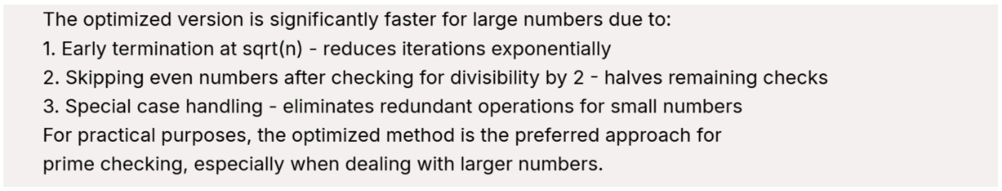


```python
def is_prime_basic(n):
    """Basic prime checker - checks all numbers up to n-1"""
    if n < 2:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True


def is_prime_optimized(n):
    """Optimized prime checker - checks up to sqrt(n) only"""
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Only check odd divisors up to sqrt(n)
    i = 3
```



```
2: basic=True, optimized=True
17: basic=True, optimized=True
100: basic=False, optimized=False
97: basic=True, optimized=True
1000: basic=False, optimized=False
```

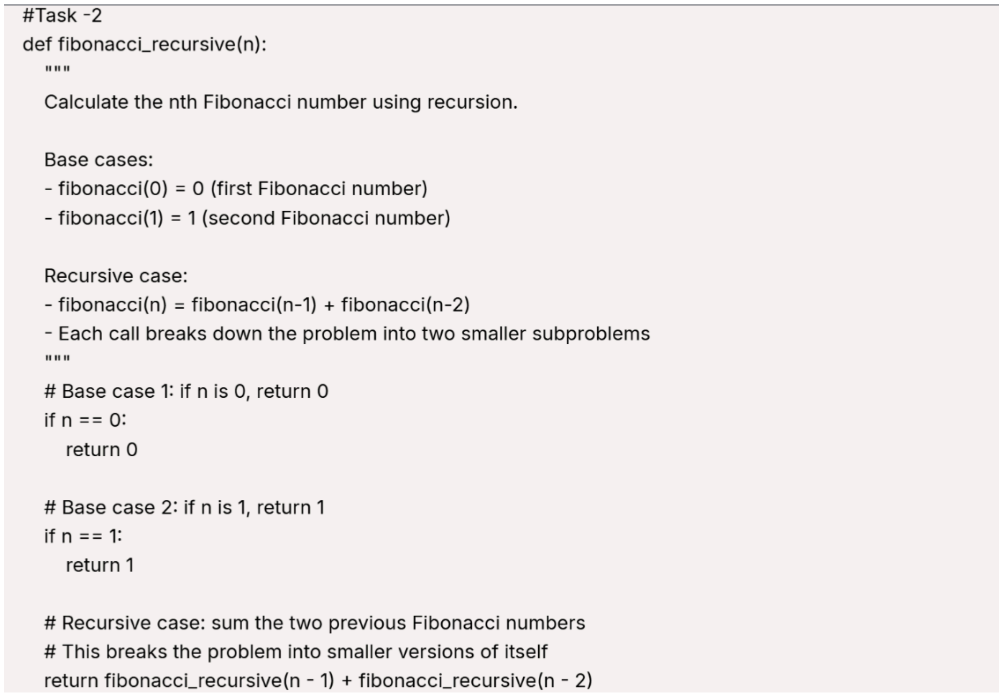
```
Prime Number Checker Module
This module provides two implementations for checking whether a number is prime,
demonstrating the difference between a basic approach and an optimized approach.
Functions:
    is_prime_basic(n): Checks if n is prime by testing divisibility against all
                    numbers from 2 to n-1.
                    Time Complexity: O(n) - performs n-2 division operations in
                    the worst case.
    is_prime_optimized(n): Checks if n is prime by testing divisibility only against
                        odd numbers up to sqrt(n), after handling small cases.
                        Time Complexity: O(sqrt(n)) - performs approximately sqrt(n)/2
                        division operations in the worst case.
Efficiency Comparison:
    - For n=100: basic checks 98 divisors, optimized checks ~5 divisors
    - For n=1000: basic checks 998 divisors, optimized checks ~15 divisors
    - For n=1000000: basic checks 999,998 divisors, optimized checks ~500 divisors
```



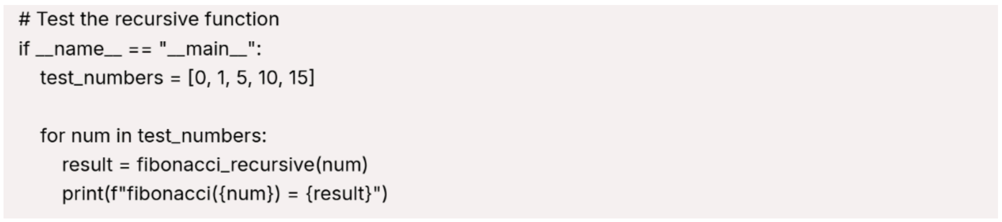
```
The optimized version is significantly faster for large numbers due to:
1. Early termination at sqrt(n) - reduces iterations exponentially
2. Skipping even numbers after checking for divisibility by 2 - halves remaining checks
3. Special case handling - eliminates redundant operations for small numbers
For practical purposes, the optimized method is the preferred approach for
prime checking, especially when dealing with larger numbers.
```

Task Description #2 (Transparency in Recursive Algorithms)
Objective: Use AI to generate a recursive function to calculate
Fibonacci numbers.
Instructions:

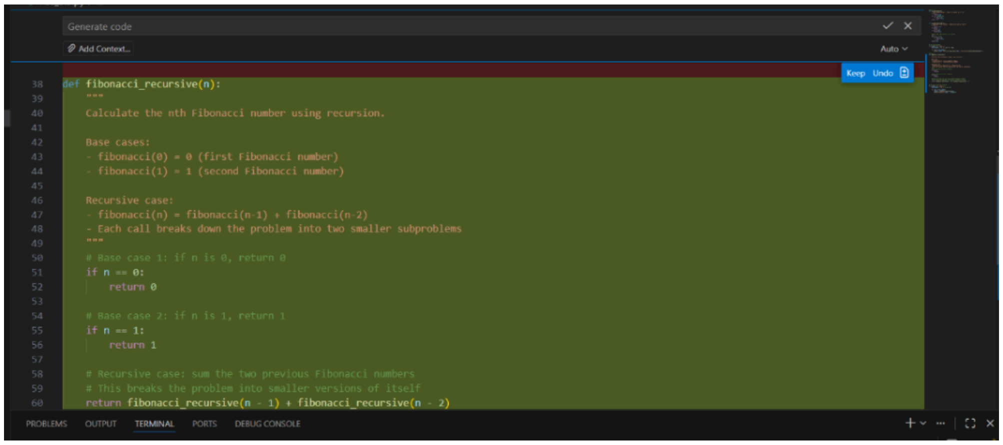
1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.
   Expected Output:
   - Well-commented recursive code.
   - Clear explanation of how recursion works.
   - Verification that explanation matches actual execution.



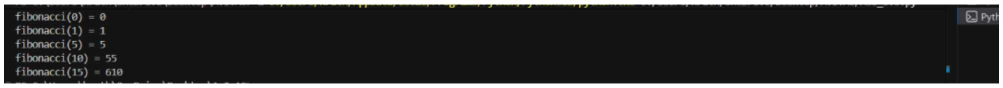
```
#Task -2
def fibonacci_recursive(n):
    """
    Calculate the nth Fibonacci number using recursion.

    Base cases:
    - fibonacci(0) = 0 (first Fibonacci number)
    - fibonacci(1) = 1 (second Fibonacci number)

    Recursive case:
    - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
    - Each call breaks down the problem into two smaller subproblems
    """
    # Base case 1: if n is 0, return 0
    if n == 0:
        return 0

    # Base case 2: if n is 1, return 1
    if n == 1:
        return 1

    # Recursive case: sum the two previous Fibonacci numbers
    # This breaks the problem into smaller versions of itself
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```



```
# Test the recursive function
if __name__ == "__main__":
    test_numbers = [0, 1, 5, 10, 15]

    for num in test_numbers:
        result = fibonacci_recursive(num)
        print(f"fibonacci({num}) = {result}")
```



```python
def fibonacci_recursive(n):
    """
    Calculate the nth Fibonacci number using recursion.

    Base cases:
    - fibonacci(0) = 0 (first Fibonacci number)
    - fibonacci(1) = 1 (second Fibonacci number)

    Recursive case:
    - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
    - Each call breaks down the problem into two smaller subproblems
    """
    # Base case 1: if n is 0, return 0
    if n == 0:
        return 0

    # Base case 2: if n is 1, return 1
    if n == 1:
        return 1

    # Recursive case: sum the two previous Fibonacci numbers
    # This breaks the problem into smaller versions of itself
    return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
```



```
fibonacci(0) = 0
fibonacci(1) = 1
fibonacci(5) = 5
fibonacci(10) = 55
fibonacci(15) = 610
```

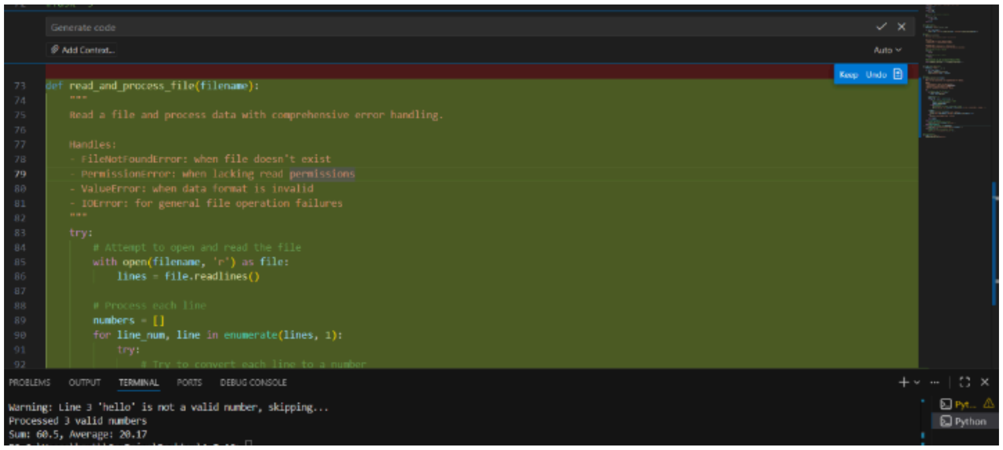
```
Factorial is defined as:
    - factorial(0) = 1 (base case: 0! = 1)
    - factorial(n) = n * factorial(n-1) for n > 0

    How recursion works here:
    1. Each call multiplies n by the result of factorial(n-1)
    2. The recursion stops when n reaches 0 (base case)
    3. Results "unwind" back up the call stack, multiplying at each level

    Example: factorial(4)
    - factorial(4) calls 4 * factorial(3)
    - factorial(3) calls 3 * factorial(2)
    - factorial(2) calls 2 * factorial(1)
    - factorial(1) calls 1 * factorial(0)
    - factorial(0) returns 1 (base case - no more recursion)
    - Then: 1*1=1, 2*1=2, 3*2=6, 4*6=24
```

Task Description #3 (Transparency in Error Handling)
Task: Use AI to generate a Python program that reads a file and processes data.
Prompt:
"Generate code with proper error handling and clear explanations for each exception."
Expected Output:
• Code with meaningful exception handling.
• Clear comments explaining each error scenario.
• Validation that explanations align with runtime behavior.

```
#Task -3
def read_and_process_file(filename):
    """
    Read a file and process data with comprehensive error handling.

    Handles:
    - FileNotFoundError: when file doesn't exist
    - PermissionError: when lacking read permissions
    - ValueError: when data format is invalid
    - IOError: for general file operation failures
    """
    try:
        # Attempt to open and read the file
        with open(filename, 'r') as file:
            lines = file.readlines()

        # Process each line
        numbers = []
        for line_num, line in enumerate(lines, 1):
            try:
                # Try to convert each line to a number
                number = float(line.strip())
                numbers.append(number)
            except ValueError:
```

```
                # Raised when line cannot be converted to float
                print(f"Warning: Line {line_num} '{line.strip()}' is not a valid number, skipping...")

        # Calculate and display statistics
        if numbers:
            print(f"Processed {len(numbers)} valid numbers")
            print(f"Sum: {sum(numbers)}, Average: {sum(numbers)/len(numbers):.2f}")
        else:
            print("No valid numbers found in file")

        return numbers
```

```
    except FileNotFoundError:
        # Raised when file path doesn't exist
        print(f"Error: File '{filename}' not found.")
    except PermissionError:
        # Raised when lacking read permissions
        print(f"Error: Permission denied reading '{filename}'.")
    except IOError as e:
        # Raised for other file operation failures
        print(f"Error: File operation failed - {e}")


# Test the function
if __name__ == "__main__":
    read_and_process_file("data.txt")
```



```python
def read_and_process_file(filename):
    """
    Read a file and process data with comprehensive error handling.

    Handles:
    - FileNotFoundError: when file doesn't exist
    - PermissionError: when lacking read permissions
    - ValueError: when data format is invalid
    - IOError: for general file operation failures
    """
    try:
        # Attempt to open and read the file
        with open(filename, 'r') as file:
            lines = file.readlines()

        # Process each line
        numbers = []
        for line_num, line in enumerate(lines, 1):
            try:
                # Try to convert each line to a number
```

```
Warning: Line 3 'hello' is not a valid number, skipping...
Processed 3 valid numbers
Sum: 60.5, Average: 20.17
```

▼ data.txt

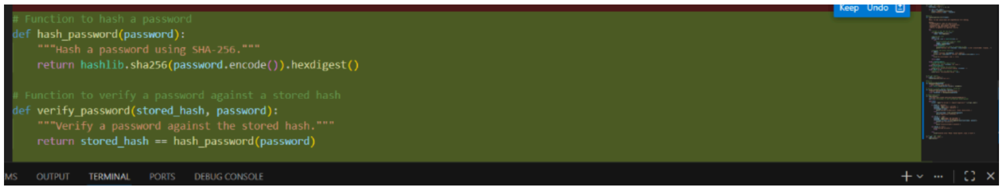
```
10
20
hello
30.5
```

Task Description #4 (Security in User Authentication)
Task: Use an AI tool to generate a Python-based login system.
Analyze: Check whether the AI uses secure password handling practices.
Expected Output:
• Identification of security flaws (plain-text passwords, weak validation).
• Revised version using password hashing and input validation.
• Short note on best practices for secure authentication.

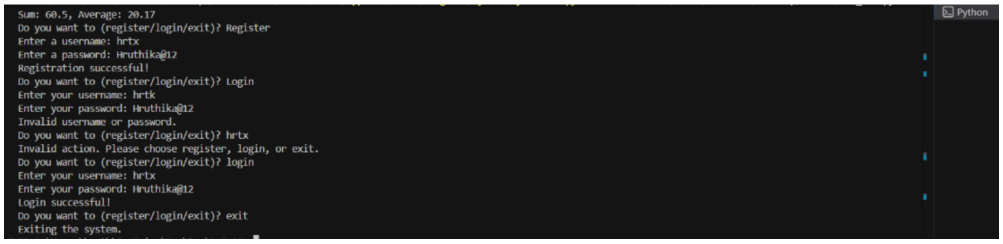
```python
#Task -4
# Function to hash a password
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()

# Function to verify a password against a stored hash
def verify_password(stored_hash, password):
    """Verify a password against the stored hash."""
    return stored_hash == hash_password(password)
```

```
# Simple login system
def login_system():
    """A simple login system with secure password handling."""
    users = {}  # Dictionary to store username and hashed password

    while True:
        action = input("Do you want to (register/login/exit)? ").strip().lower()

        if action == 'register':
            username = input("Enter a username: ")
            password = input("Enter a password: ")
            if username in users:
                print("Username already exists. Please choose another.")
            else:
                users[username] = hash_password(password)
                print("Registration successful!")

        elif action == 'login':
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            if username in users and verify_password(users[username], password):
                print("Login successful!")
            else:
                print("Invalid username or password.")
```

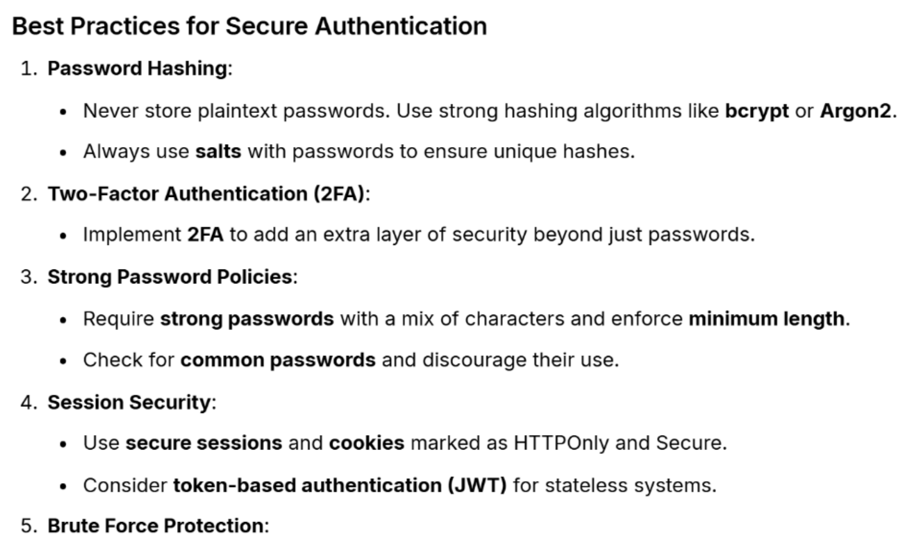
```
        elif action == 'exit':
            print("Exiting the system.")
            break

        else:
            print("Invalid action. Please choose register, login, or exit.")

# Test the login system
if __name__ == "__main__":
    login_system()
```



```
# Function to hash a password
def hash_password(password):
    """Hash a password using SHA-256."""
    return hashlib.sha256(password.encode()).hexdigest()

# Function to verify a password against a stored hash
def verify_password(stored_hash, password):
    """Verify a password against the stored hash."""
    return stored_hash == hash_password(password)
```

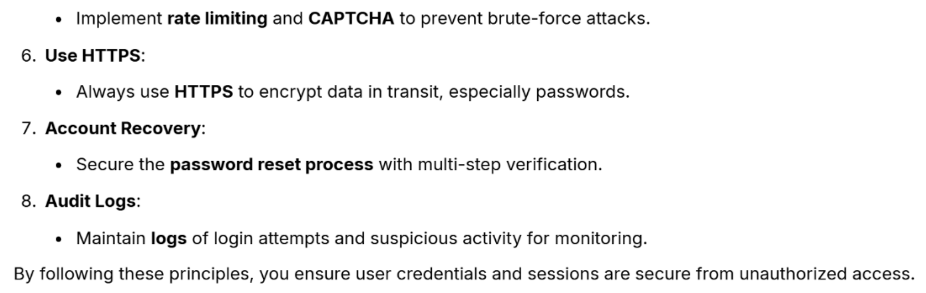


```
Sum: 60.5, Average: 20.17
Do you want to (register/login/exit)? Register
Enter a username: hrtx
Enter a password: Hruthika@12
Registration successful!
Do you want to (register/login/exit)? Login
Enter your username: hrtk
Enter your password: Hruthika@12
Invalid username or password.
Do you want to (register/login/exit)? hrtx
Invalid action. Please choose register, login, or exit.
Do you want to (register/login/exit)? login
Enter your username: hrtx
Enter your password: Hruthika@12
Login successful!
Do you want to (register/login/exit)? exit
Exiting the system.
```



Best Practices for Secure Authentication

1. **Password Hashing**:
   - Never store plaintext passwords. Use strong hashing algorithms like **bcrypt** or **Argon2**.
   - Always use **salts** with passwords to ensure unique hashes.
2. **Two-Factor Authentication (2FA)**:
   - Implement **2FA** to add an extra layer of security beyond just passwords.
3. **Strong Password Policies**:
   - Require **strong passwords** with a mix of characters and enforce **minimum length**.
   - Check for **common passwords** and discourage their use.
4. **Session Security**:
   - Use **secure sessions** and **cookies** marked as HTTPOnly and Secure.
   - Consider **token-based authentication (JWT)** for stateless systems.
5. **Brute Force Protection**:



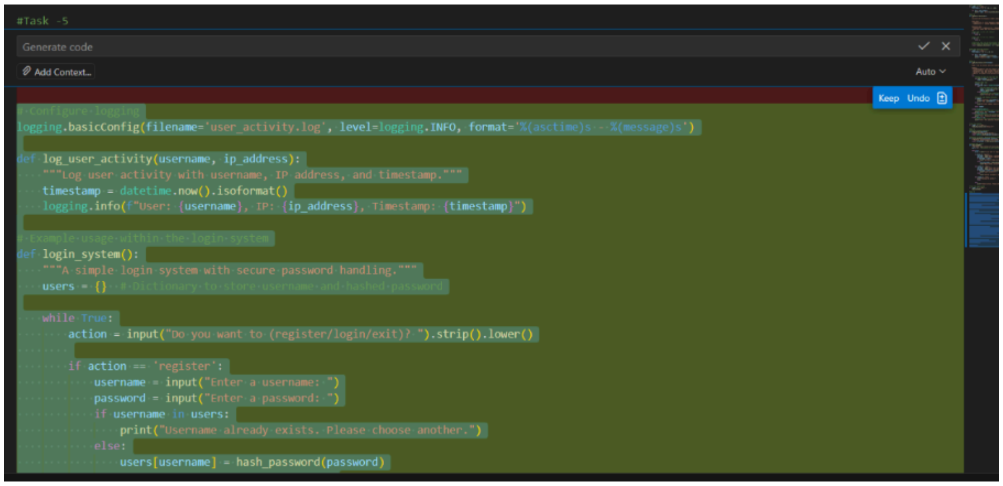
   - Implement **rate limiting** and **CAPTCHA** to prevent brute-force attacks.
6. **Use HTTPS**:
   - Always use **HTTPS** to encrypt data in transit, especially passwords.
7. **Account Recovery**:
   - Secure the **password reset process** with multi-step verification.
8. **Audit Logs**:
   - Maintain **logs** of login attempts and suspicious activity for monitoring.

By following these principles, you ensure user credentials and sessions are secure from unauthorized access.

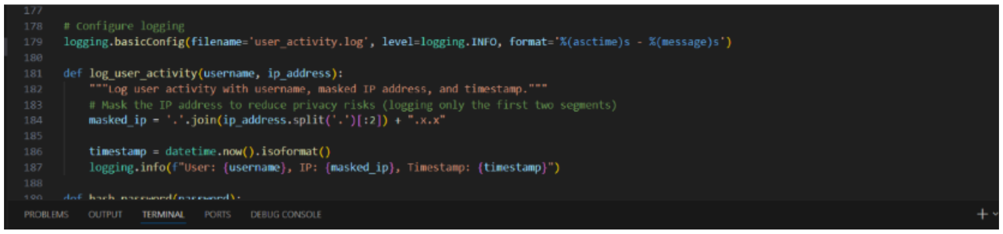
Task Description #5 (Privacy in Data Logging)
Task: Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).
Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.
Expected Output:
• Identified privacy risks in logging.
• Improved version with minimal, anonymized, or masked logging.
• Explanation of privacy-aware logging principles.

```
#Task -5
import logging
from datetime import datetime
import hashlib

# Configure logging
logging.basicConfig(filename='user_activity.log', level=logging.INFO, format='%(asctime)s - %(message)
s')

def log_user_activity(username, ip_address):
    """Log user activity with username, masked IP address, and timestamp."""
    # Mask the IP address to reduce privacy risks (logging only the first two segments)
    masked_ip = '.'.join(ip_address.split('.')[:2]) + ".x.x"

    timestamp = datetime.now().isoformat()
    logging.info(f"User: {username}, IP: {masked_ip}, Timestamp: {timestamp}")

def hash_password(password):
    """Hash the password for secure storage."""
    # Using a simple hash function for demonstration; in a real system, use a secure hashing algorithm like
bcrypt
    return hashlib.sha256(password.encode()).hexdigest()
```

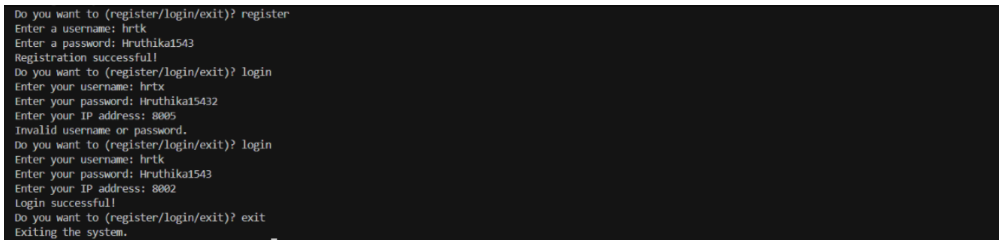
```
def verify_password(hashed_password, input_password):
    """Verify the entered password against the stored hash."""
    return hashed_password == hash_password(input_password)

# Example usage within the login system
def login_system():
    """A simple login system with secure password handling."""
    users = {}  # Dictionary to store username and hashed password

    while True:
        action = input("Do you want to (register/login/exit)? ").strip().lower()

        if action == 'register':
            username = input("Enter a username: ")
            password = input("Enter a password: ")
            if username in users:
                print("Username already exists. Please choose another.")
            else:
                users[username] = hash_password(password)
                print("Registration successful!")
        elif action == 'login':
            username = input("Enter your username: ")
            password = input("Enter your password: ")
            ip_address = input("Enter your IP address: ")  # Simulated input for demonstration
            if username in users and verify_password(users[username], password):
                print("Login successful!")
                log_user_activity(username, ip_address)  # Log the activity with masked IP
            else:
                print("Invalid username or password.")

        elif action == 'exit':
            print("Exiting the system.")
            break

        else:
            print("Invalid action. Please choose register, login, or exit.")

if __name__ == "__main__":
    login_system()
```



```
#Task -5
# Configure logging
logging.basicConfig(filename='user_activity.log', level=logging.INFO, format='%(asctime)s - %(message)s')

def log_user_activity(username, ip_address):
    """Log user activity with username, IP address, and timestamp."""
    timestamp = datetime.now().isoformat()
    logging.info(f"User: {username}, IP: {ip_address}, Timestamp: {timestamp}")

# Example usage within the login system
def login_system():
    """A simple login system with secure password handling."""
    users = {}  # Dictionary to store username and hashed password

    while True:
        action = input("Do you want to (register/login/exit)? ").strip().lower()

        if action == 'register':
            username = input("Enter a username: ")
            password = input("Enter a password: ")
            if username in users:
                print("Username already exists. Please choose another.")
            else:
                users[username] = hash_password(password)
```



```
# Configure logging
logging.basicConfig(filename='user_activity.log', level=logging.INFO, format='%(asctime)s - %(message)s')

def log_user_activity(username, ip_address):
    """Log user activity with username, masked IP address, and timestamp."""
    # Mask the IP address to reduce privacy risks (logging only the first two segments)
    masked_ip = '.'.join(ip_address.split('.')[:2]) + ".x.x"

    timestamp = datetime.now().isoformat()
    logging.info(f"User: {username}, IP: {masked_ip}, Timestamp: {timestamp}")

def hash_password(password):
```



```
Do you want to (register/login/exit)? register
Enter a username: hrtk
Enter a password: Hruthika1543
Registration successful!
Do you want to (register/login/exit)? login
Enter your username: hrtx
Enter your password: Hruthika15432
Enter your IP address: 8005
Invalid username or password.
Do you want to (register/login/exit)? login
Enter your username: hrtk
Enter your password: Hruthika1543
Enter your IP address: 8002
Login successful!
Do you want to (register/login/exit)? exit
Exiting the system.
```

Privacy-Aware Logging Principles

1. **Minimize Data Collection**:
   - Only log necessary data and avoid sensitive information like passwords. Mask or anonymize data (e.g., logging partial IP addresses).
2. **Masking and Anonymization**:
   - Mask sensitive data (like IP addresses) and avoid logging full details. Use hashing for sensitive information like passwords.
3. **Secure Storage**:
   - Store logs securely, use encryption, and restrict access with proper permissions to protect sensitive data.
4. **Data Retention**:
   - Keep logs only as long as needed and delete or anonymize old logs to reduce exposure risks.
5. **User Consent**:
   - Inform users about data logging and obtain consent, especially in regions with strict privacy laws like GDPR.
6. **Audit Logs**:
   - Use logs to monitor for suspicious activity but limit access to logs and ensure they are only available to authorized personnel.
7. **Never Log Plaintext Passwords**:
   - Always hash passwords before storing or verifying them, never logging them in plaintext.