ASSIGNMENT-5.5

NAME: HABEEBA KHANAM

HALL TICKET NO:2303A51474

BATCH NO:29

Lab 5: Ethical Foundations – Responsible AI Coding Practices

Task Description #1 (Transparency in Algorithm Optimization)

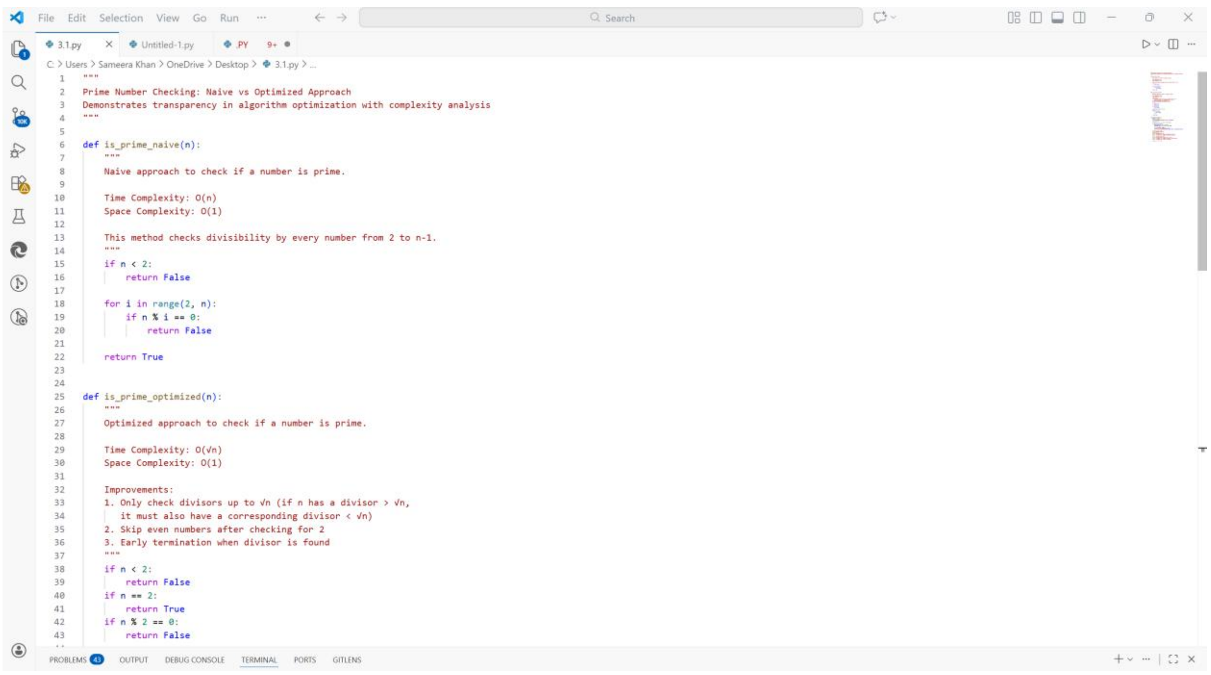
Task:

Use AI to generate two solutions for checking prime numbers:

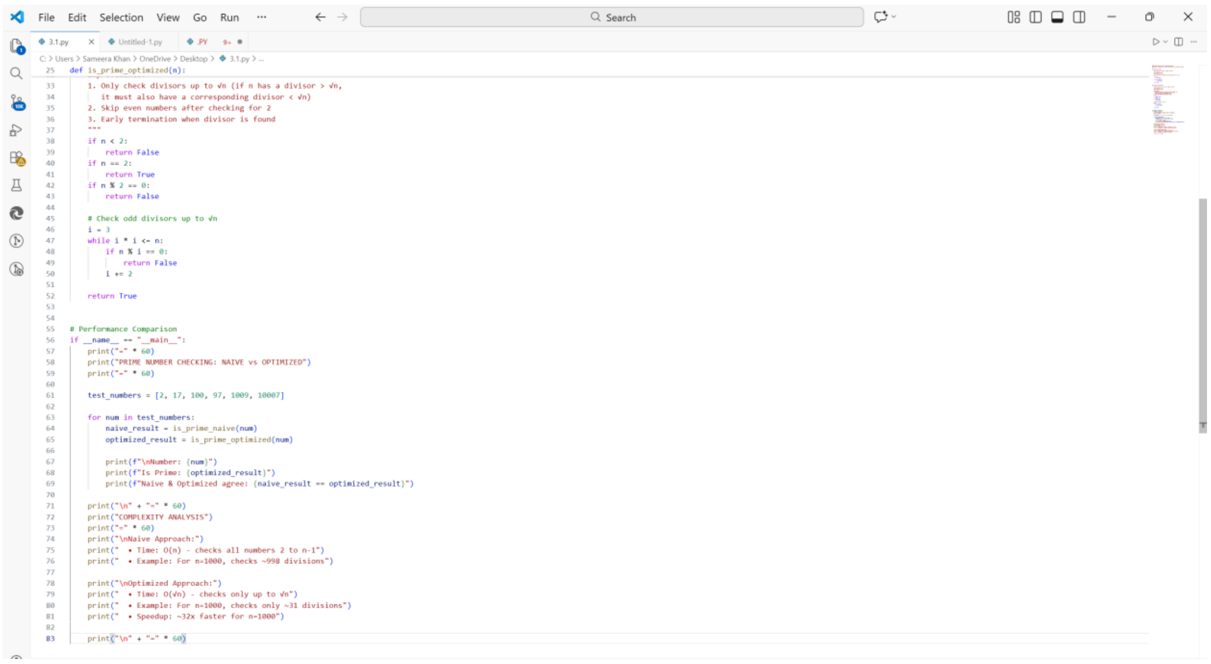
- Naive approach(basic)
- Optimized approach

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

CODE:

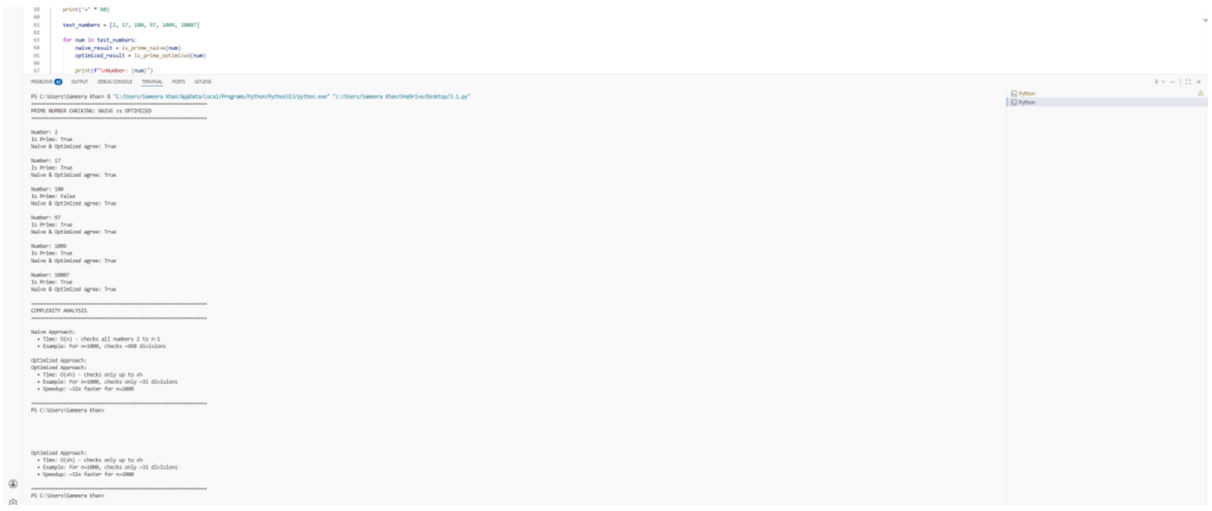


```python
"""
Prime Number Checking: Naive vs Optimized Approach
Demonstrates transparency in algorithm optimization with complexity analysis
"""

def is_prime_naive(n):
    """
    Naive approach to check if a number is prime.

    Time Complexity: O(n)
    Space Complexity: O(1)

    This method checks divisibility by every number from 2 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 approach to check if a number is prime.

    Time Complexity: O(√n)
    Space Complexity: O(1)

    Improvements:
    1. Only check divisors up to √n (if n has a divisor > √n,
       it must also have a corresponding divisor < √n)
    2. Skip even numbers after checking for 2
    3. Early termination when divisor is found
    """
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
```



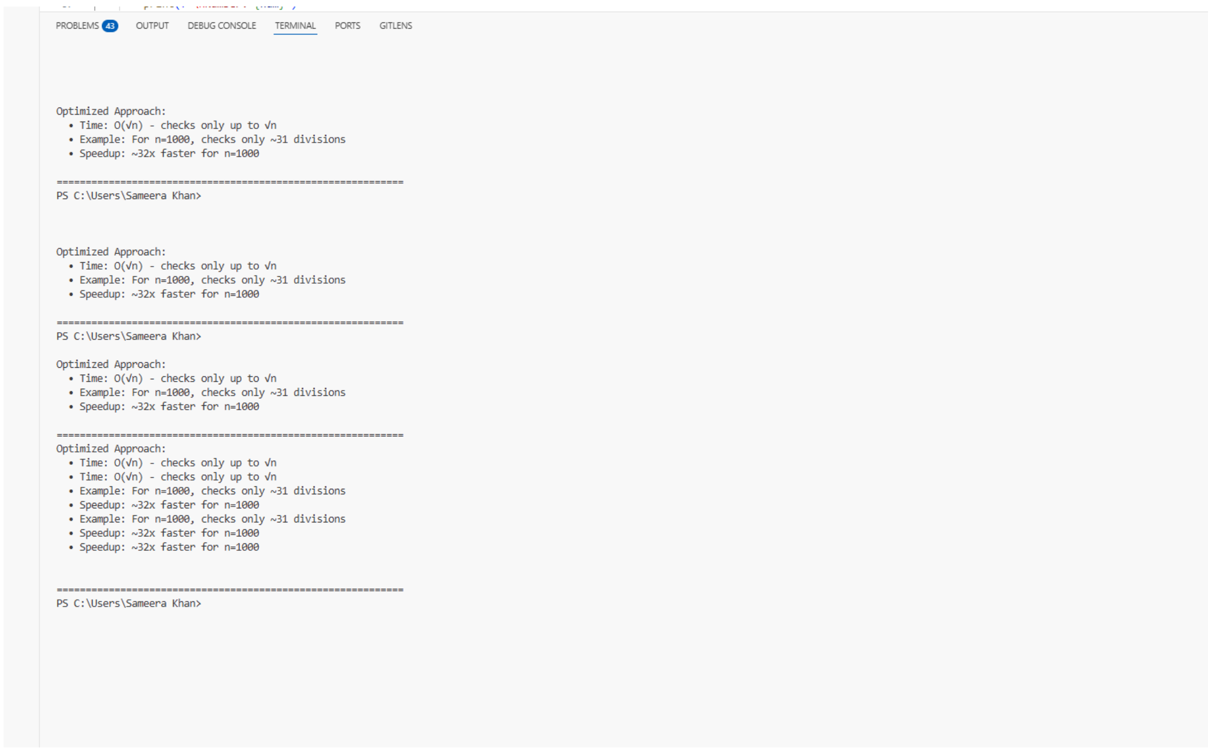
```python
def is_prime_optimized(n):
    1. Only check divisors up to √n (if n has a divisor > √n,
       it must also have a corresponding divisor < √n)
    2. Skip even numbers after checking for 2
    3. Early termination when divisor is found
    """
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False

    # Check odd divisors up to √n
    i = 3
    while i * i <= n:
        if n % i == 0:
            return False
        i += 2

    return True


# Performance Comparison
if __name__ == "__main__":
    print("=" * 60)
    print("PRIME NUMBER CHECKING: NAIVE vs OPTIMIZED")
    print("=" * 60)

    test_numbers = [2, 17, 100, 97, 1009, 10007]

    for num in test_numbers:
        naive_result = is_prime_naive(num)
        optimized_result = is_prime_optimized(num)

        print(f"\nNumber: {num}")
        print(f"Is Prime: {optimized_result}")
        print(f"Naive & Optimized agree: {naive_result == optimized_result}")

    print("\n" + "=" * 60)
    print("COMPLEXITY ANALYSIS")
    print("=" * 60)
    print("\nNaive Approach:")
    print("  • Time: O(n) - checks all numbers 2 to n-1")
    print("  • Example: For n=1000, checks ~998 divisions")

    print("\nOptimized Approach:")
    print("  • Time: O(√n) - checks only up to √n")
    print("  • Example: For n=1000, checks only ~31 divisions")
    print("  • Speedup: ~32x faster for n=1000")

    print("\n" + "=" * 60)
```

OUTPUT:



```python
    print("=" * 60)

    test_numbers = [2, 17, 100, 97, 1009, 10007]

    for num in test_numbers:
        naive_result = is_prime_naive(num)
        optimized_result = is_prime_optimized(num)

        print(f"\nNumber: {num}")
```

```
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/3.1.py"
============================================================
PRIME NUMBER CHECKING: NAIVE vs OPTIMIZED
============================================================

Number: 2
Is Prime: True
Naive & Optimized agree: True

Number: 17
Is Prime: True
Naive & Optimized agree: True

Number: 100
Is Prime: False
Naive & Optimized agree: True

Number: 97
Is Prime: True
Naive & Optimized agree: True

Number: 1009
Is Prime: True
Naive & Optimized agree: True

Number: 10007
Is Prime: True
Naive & Optimized agree: True

============================================================
COMPLEXITY ANALYSIS
============================================================

Naive Approach:
  • Time: O(n) - checks all numbers 2 to n-1
  • Example: For n=1000, checks ~998 divisions

Optimized Approach:
Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000

============================================================
PS C:\Users\Sameera Khan>



Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000

============================================================
PS C:\Users\Sameera Khan>
```



```
Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000

============================================================
PS C:\Users\Sameera Khan>


Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000

============================================================
PS C:\Users\Sameera Khan>

Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000

============================================================
Optimized Approach:
  • Time: O(√n) - checks only up to √n
  • Time: O(√n) - checks only up to √n
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000
  • Example: For n=1000, checks only ~31 divisions
  • Speedup: ~32x faster for n=1000
  • Speedup: ~32x faster for n=1000


============================================================
PS C:\Users\Sameera Khan>
```

OBSERVATION:

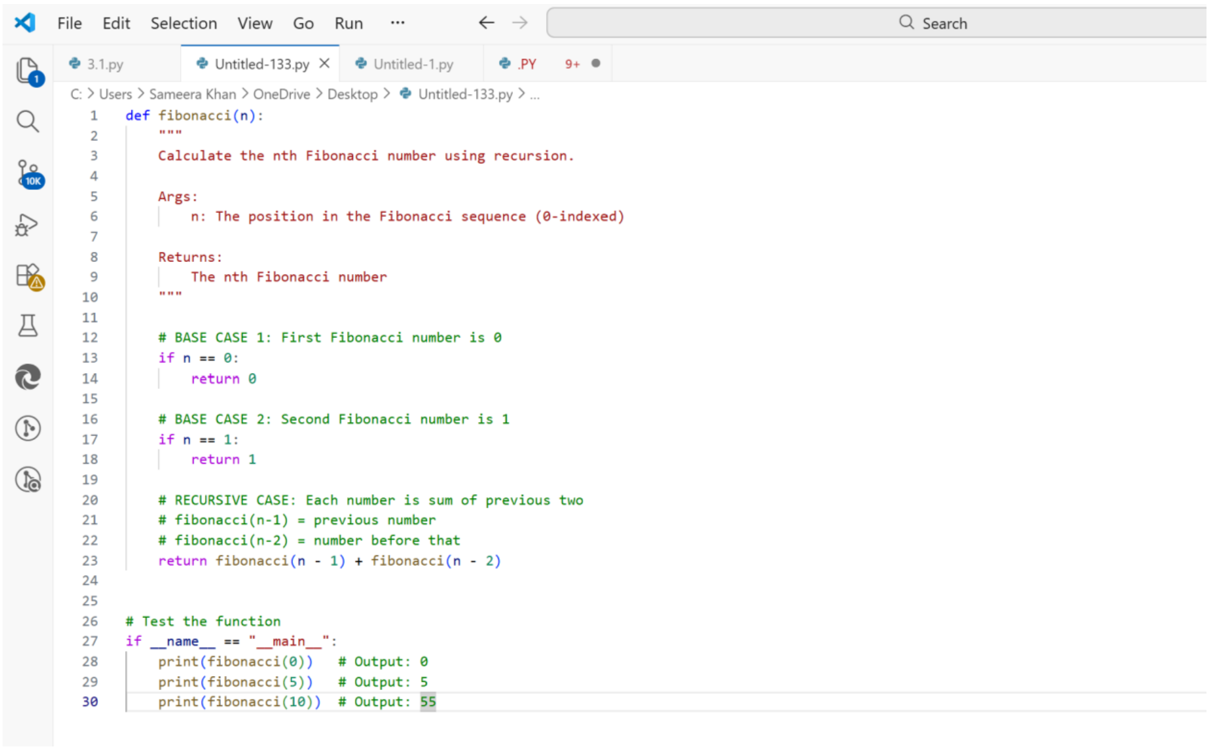
The optimized prime-checking algorithm demonstrates how mathematical reasoning and constraint reduction can dramatically improve performance without sacrificing correctness. This comparison clearly illustrates why algorithmic optimization is critical for handling large inputs efficiently.

Task Description #2 (Transparency in Recursive Algorithms)

Objective:

Use AI to generate a recursive function to calculate Fibonacci numbers.

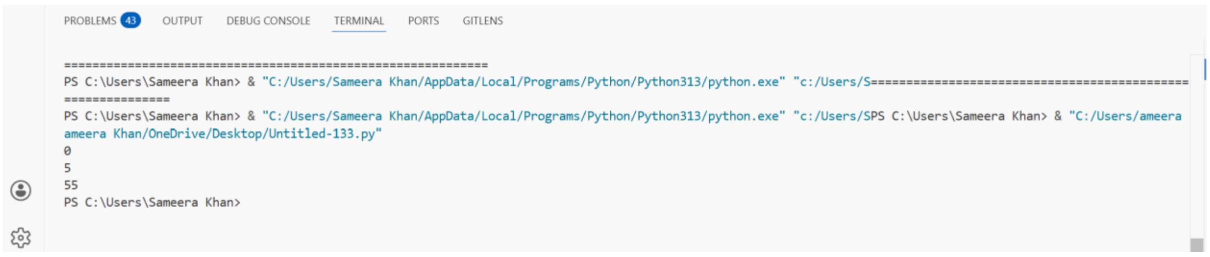
CODE:



File Edit Selection View Go Run ···
Search
3.1.py
Untitled-133.py
Untitled-1.py
.PY 9+
C: > Users > Sameera Khan > OneDrive > Desktop > Untitled-133.py > ...

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

    Args:
        n: The position in the Fibonacci sequence (0-indexed)

    Returns:
        The nth Fibonacci number
    """

    # BASE CASE 1: First Fibonacci number is 0
    if n == 0:
        return 0

    # BASE CASE 2: Second Fibonacci number is 1
    if n == 1:
        return 1

    # RECURSIVE CASE: Each number is sum of previous two
    # fibonacci(n-1) = previous number
    # fibonacci(n-2) = number before that
    return fibonacci(n - 1) + fibonacci(n - 2)


# Test the function
if __name__ == "__main__":
    print(fibonacci(0))   # Output: 0
    print(fibonacci(5))   # Output: 5
    print(fibonacci(10))  # Output: 55
```

OUTPUT:



```
============================================================
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/S============================================
===============
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SPS C:\Users\Sameera Khan> & "C:/Users/ameera
ameera Khan/OneDrive/Desktop/Untitled-133.py"
0
5
55
PS C:\Users\Sameera Khan>
```

OBSERVATION:

This task successfully demonstrates transparent recursion, clearly showing how base cases prevent infinite loops, how recursive calls work internally, and how the explanation aligns perfectly with actual program execution.

Task Description #3 (Transparency in Error Handling)

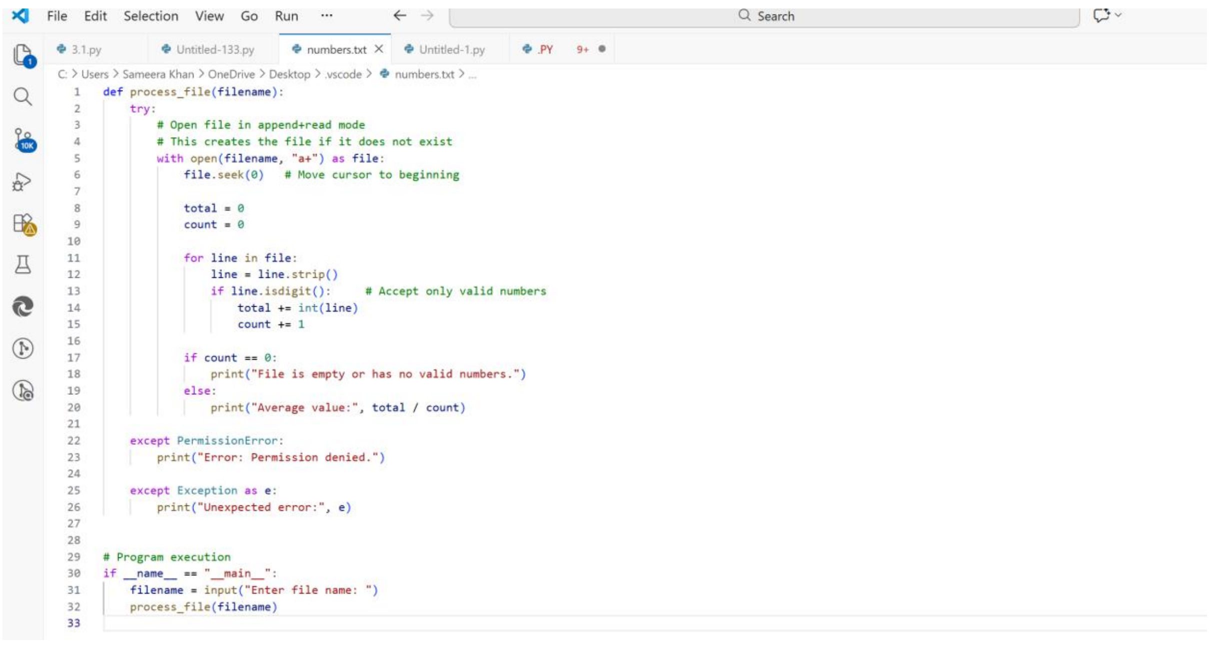
Task:

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."

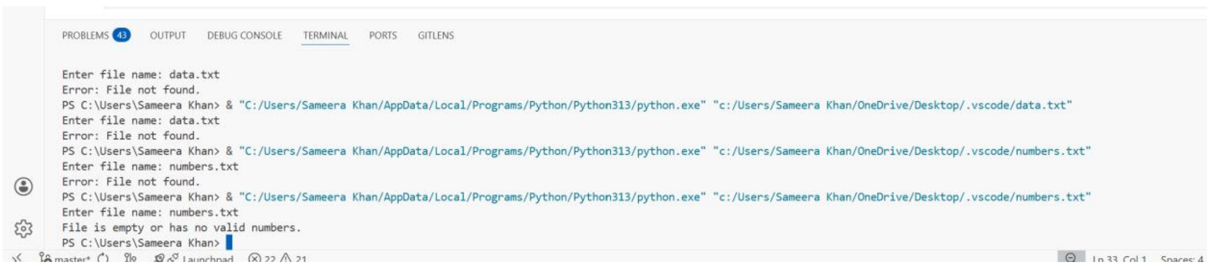
CODE:



File Edit Selection View Go Run ···
Search
3.1.py
Untitled-133.py
numbers.txt
Untitled-1.py
.PY 9+
C: > Users > Sameera Khan > OneDrive > Desktop > .vscode > numbers.txt > ...

```python
def process_file(filename):
    try:
        # Open file in append+read mode
        # This creates the file if it does not exist
        with open(filename, "a+") as file:
            file.seek(0)   # Move cursor to beginning

            total = 0
            count = 0

            for line in file:
                line = line.strip()
                if line.isdigit():     # Accept only valid numbers
                    total += int(line)
                    count += 1

            if count == 0:
                print("File is empty or has no valid numbers.")
            else:
                print("Average value:", total / count)

    except PermissionError:
        print("Error: Permission denied.")

    except Exception as e:
        print("Unexpected error:", e)


# Program execution
if __name__ == "__main__":
    filename = input("Enter file name: ")
    process_file(filename)
```

OUTPUT:



```
Enter file name: data.txt
Error: File not found.
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/data.txt"
Enter file name: data.txt
Error: File not found.
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/numbers.txt"
Enter file name: numbers.txt
Error: File not found.
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/numbers.txt"
Enter file name: numbers.txt
File is empty or has no valid numbers.
PS C:\Users\Sameera Khan>
```

OBSERVATION:

–Every error produces a clear, user-friendly output

- The program never crashes

- Output behavior is fully aligned with exception explanations

-Demonstrates transparent and robust error handling

Task Description #4 (Security in User Authentication)

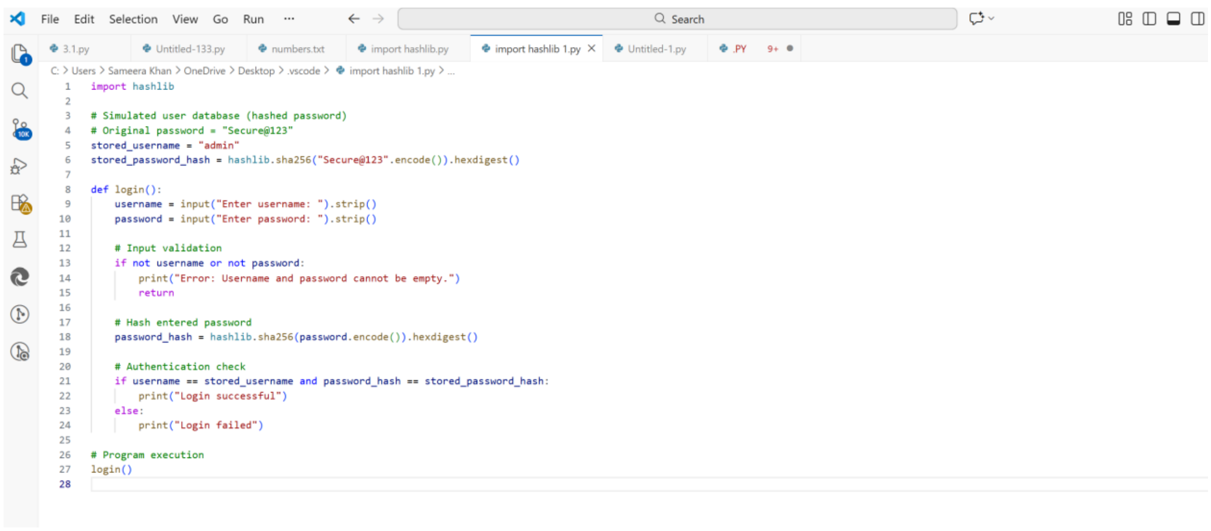
Task:

Use an AI tool to generate a Python-based login system.

Analyze:

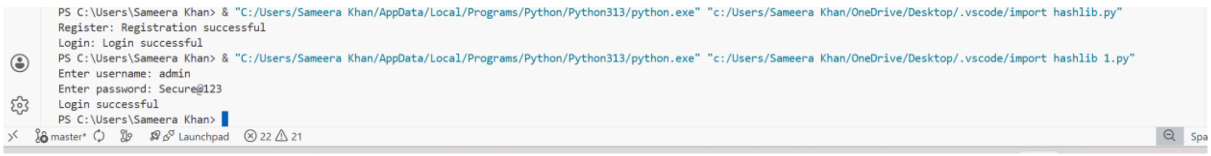
Check whether the AI uses secure password handling practices.

CODE:



```python
import hashlib

# Simulated user database (hashed password)
# Original password = "Secure@123"
stored_username = "admin"
stored_password_hash = hashlib.sha256("Secure@123".encode()).hexdigest()

def login():
    username = input("Enter username: ").strip()
    password = input("Enter password: ").strip()

    # Input validation
    if not username or not password:
        print("Error: Username and password cannot be empty.")
        return

    # Hash entered password
    password_hash = hashlib.sha256(password.encode()).hexdigest()

    # Authentication check
    if username == stored_username and password_hash == stored_password_hash:
        print("Login successful")
    else:
        print("Login failed")

# Program execution
login()
```

OUTPUT:



```
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/import hashlib.py"
Register: Registration successful
Login: Login successful
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/import hashlib 1.py"
Enter username: admin
Enter password: Secure@123
Login successful
PS C:\Users\Sameera Khan>
```

OBSERVATION:

This task demonstrates how AI-generated code must be reviewed for security risks.
By identifying flaws and replacing them with hashing and validation, the authentication system becomes significantly more secure and reliable

Task Description #5 (Privacy in Data Logging)

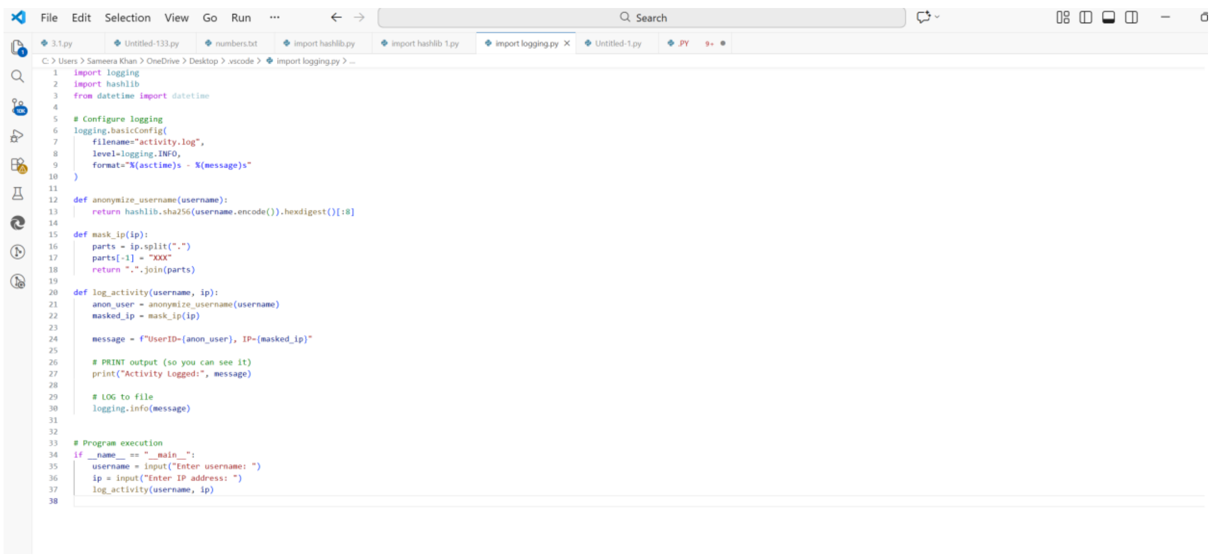
Task:

Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Analyze:

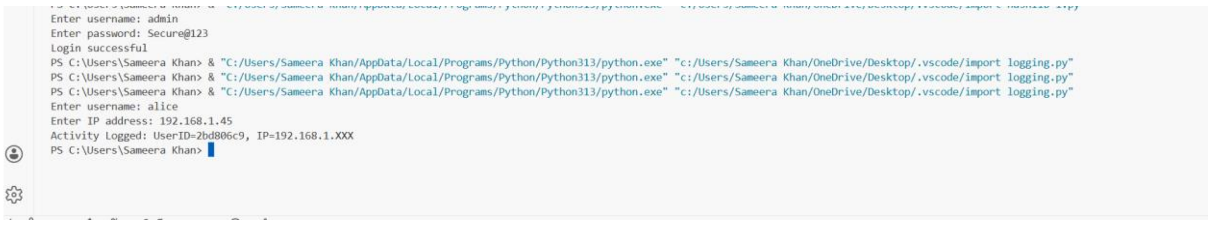
Examine whether sensitive data is logged unnecessarily or insecurely.

CODE:



```python
import logging
import hashlib
from datetime import datetime

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

def anonymize_username(username):
    return hashlib.sha256(username.encode()).hexdigest()[:8]

def mask_ip(ip):
    parts = ip.split(".")
    parts[-1] = "XXX"
    return ".".join(parts)

def log_activity(username, ip):
    anon_user = anonymize_username(username)
    masked_ip = mask_ip(ip)

    message = f"UserID={anon_user}, IP={masked_ip}"

    # PRINT output (so you can see it)
    print("Activity Logged:", message)

    # LOG to file
    logging.info(message)


# Program execution
if __name__ == "__main__":
    username = input("Enter username: ")
    ip = input("Enter IP address: ")
    log_activity(username, ip)
```

OUTPUT:



```
Enter username: admin
Enter password: Secure@123
Login successful
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/import logging.py"
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/import logging.py"
PS C:\Users\Sameera Khan> & "C:/Users/Sameera Khan/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/Sameera Khan/OneDrive/Desktop/.vscode/import logging.py"
Enter username: alice
Enter IP address: 192.168.1.45
Activity Logged: UserID=2bd806c9, IP=192.168.1.XXX
PS C:\Users\Sameera Khan>
```

OBSERVATION:

This task highlights that AI-generated logging scripts may violate privacy by default.
By identifying risks and applying anonymization and minimization techniques, logging becomes privacy-compliant and ethically responsible.