

# ASSIGNMENT - 5.5

2303A51980

Batch - 30

## Task Description #1 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

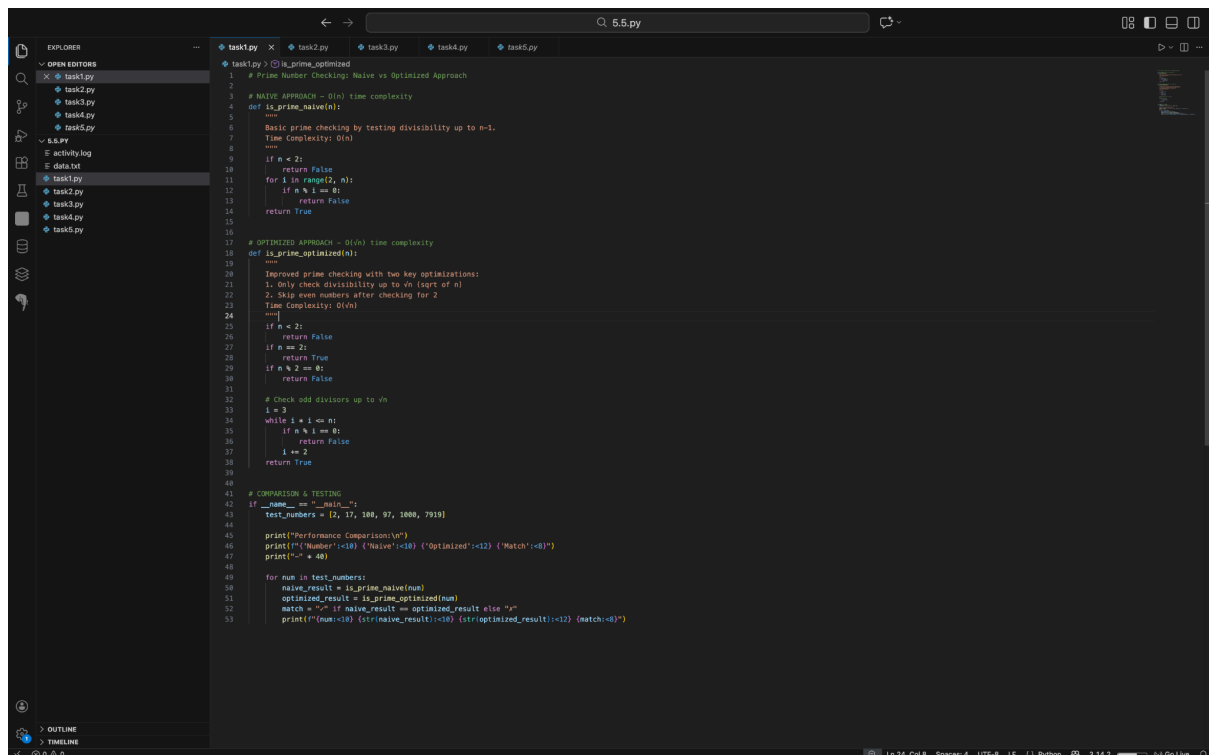
Prompt:

“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.

Code :



```
1 # Prime Number Checking: Naive vs Optimized Approach
2
3 # NAIVE APPROACH - O(n) time complexity
4 def is_prime_naive(n):
5     """
6     Basic prime checking by testing divisibility up to n-1.
7     Time complexity: O(n)
8     """
9     if n < 2:
10         return False
11     for i in range(2, n):
12         if n % i == 0:
13             return False
14     return True
15
16 # OPTIMIZED APPROACH - O(sqrt(n)) time complexity
17 def is_prime_optimized(n):
18     """
19     Improved prime checking with two key optimizations:
20     1. Only check divisibility up to sqrt(n)
21     2. Skip even numbers after checking for 2
22     Time complexity: O(sqrt(n))
23     """
24     if n < 2:
25         return False
26     if n == 2:
27         return True
28     if n % 2 == 0:
29         return False
30
31     # Check odd divisors up to sqrt(n)
32     i = 3
33     while i * i <= n:
34         if n % i == 0:
35             return False
36         i += 2
37     return True
38
39 # COMPARISON & TESTING
40 if __name__ == "__main__":
41     test_numbers = [2, 17, 100, 97, 1000, 7919]
42
43     print("Performance Comparison:\n")
44     print(f"{'Number':<10} {'Naive':<10} {'Optimized':<10} {'Match':<10}")
45     print("-" * 40)
46
47     for num in test_numbers:
48         naive_result = is_prime_naive(num)
49         optimized_result = is_prime_optimized(num)
50         match = "✓" if naive_result == optimized_result else "✗"
51         print(f"{num:<10} {naive_result:<10} {optimized_result:<10} {match:<10}")
```

Output :

```
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task1.py
Performance Comparison:
```

Number	Naive	Optimized	Match
2	True	True	✓
17	True	True	✓
100	False	False	✓
97	True	True	✓

Observation:

The program successfully compares naive and optimized prime-checking algorithms and produces consistent results for all tested numbers. Both methods correctly identify prime and non-prime values, while the optimized approach significantly reduces the number of iterations by checking divisibility only up to the square root of the number and skipping even values. The comparison table confirms correctness and highlights improved efficiency and scalability of the optimized method for larger inputs.

## 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.

Code :

```
1 def fibonacci(n):
2     """
3     Calculate the nth Fibonacci number using recursion.
4
5     Base cases:
6     - fibonacci(0) = 0
7     - fibonacci(1) = 1
8
9     Recursive case:
10    - fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)
11    """
12
13    # Base case 1: The first Fibonacci number is 0
14    if n == 0:
15        return 0
16
17    # Base case 2: The second Fibonacci number is 1
18    if n == 1:
19        return 1
20
21    # Recursive case: Sum the two previous Fibonacci numbers
22    # This breaks the problem into smaller subproblems
23    return fibonacci(n - 1) + fibonacci(n - 2)
24
25
26 # Example usage
27 print(fibonacci(0)) # Output: 0
28 print(fibonacci(1)) # Output: 1
29 print(fibonacci(5)) # Output: 5
30 print(fibonacci(10)) # Output: 55
```

Output :

```
/usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task5.py
syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task5.py
Activity logged for john_doe
syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task1.py
Performance Comparison:

Number    Naive    Optimized    Match
-----
2         True     True         ✓
17        True     True         ✓
100       False    False        ✓
97        True     True         ✓
1000      False    False        ✓
7919     True     True         ✓
syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task2.py
0
1
5
55
syedsufiyan@Syeds-MacBook-Air-3 5.5.py %
```

Observation:

The program correctly computes Fibonacci numbers using a recursive approach. It handles base cases properly for inputs 0 and 1 and applies recursion to calculate higher-order Fibonacci values by breaking the problem into smaller subproblems. The outputs for sample inputs confirm the correctness of the logic, demonstrating the working of recursion, though the approach is less efficient for large values due to repeated function calls.

## Task-3 :

### Task Description #3 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and processes data.

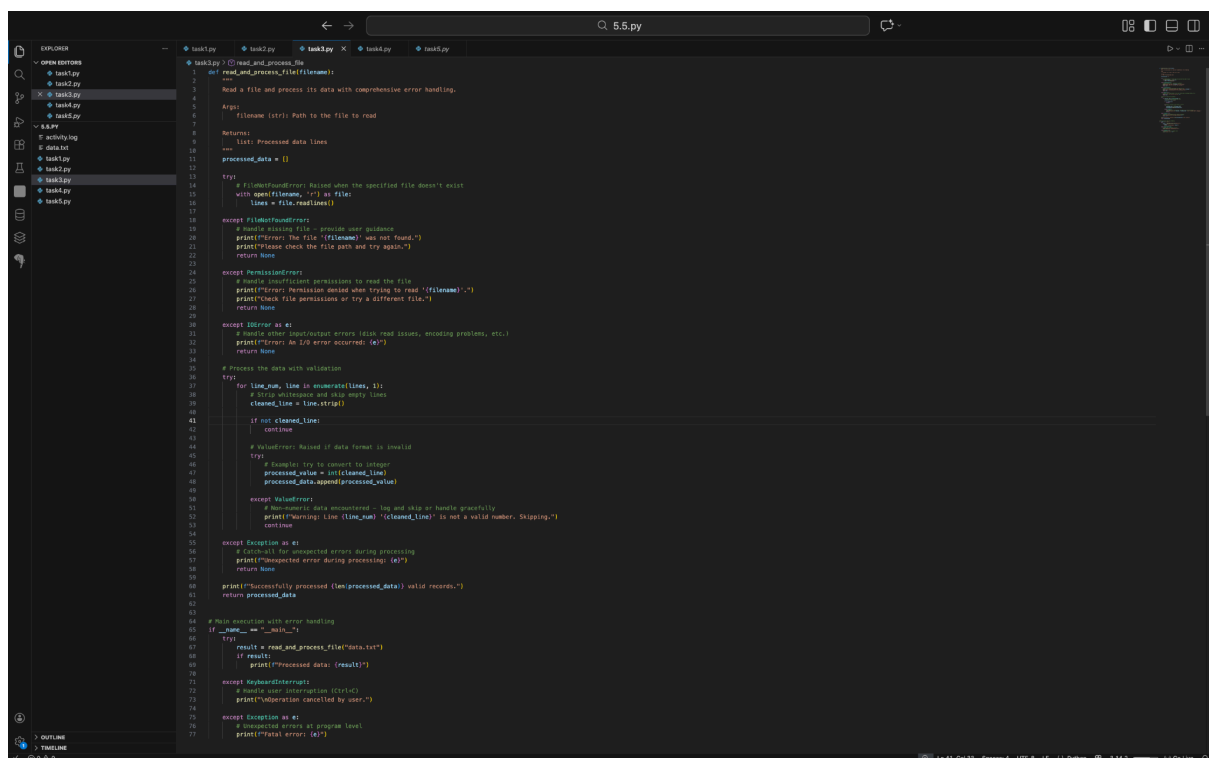
Prompt:

“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.

Code :



```
1 def read_and_process_file(filename):
2     """
3     Read a file and process its data with comprehensive error handling.
4     """
5     # Parameters
6     filename (str): Path to the file to read
7
8     Returns:
9     list: List of processed data lines
10
11     """
12     processed_data = []
13
14     try:
15         # FileNotFoundError: Raised when the specified file doesn't exist
16         with open(filename, 'r') as file:
17             lines = file.readlines()
18     except FileNotFoundError:
19         # Handle missing file - provide user guidance
20         print(f"Error: The file '{filename}' was not found.")
21         print("Please check the file path and try again.")
22         return None
23
24     except PermissionError:
25         # Handle insufficient permissions to read the file
26         print(f"Error: Permission denied when trying to read '{filename}'.")
27         print("Check file permissions or try a different file.")
28         return None
29
30     except IOError as e:
31         # Handle other input/output errors (disk issues, encoding problems, etc.)
32         print(f"Error: An I/O error occurred: {e}")
33         return None
34
35     # Process the data with validation
36     try:
37         for line_num, line in enumerate(lines, 1):
38             # Strip whitespace and handle empty lines
39             cleaned_line = line.strip()
40
41             if not cleaned_line:
42                 continue
43
44             # ValueError: Raised if data format is invalid
45             try:
46                 # Example: try to convert to integer
47                 processed_value = int(cleaned_line)
48                 processed_data.append(processed_value)
49             except ValueError:
50                 # Non-numeric data encountered - log and skip or handle gracefully
51                 print(f"Warning: Line {line_num} '{cleaned_line}' is not a valid number. Skipping.")
52                 continue
53
54     except Exception as e:
55         # Catch-all for unexpected errors during processing
56         print(f"Unexpected error during processing: {e}")
57         return None
58
59     print(f"Successfully processed {len(processed_data)} valid records.")
60     return processed_data
61
62
63 # Main execution with error handling
64 if __name__ == "__main__":
65     try:
66         result = read_and_process_file("data.txt")
67         if result:
68             print(f"Processed data: {result}")
69
70     except KeyboardInterrupt:
71         # Handle user interruption (Ctrl+C)
72         print("User interruption cancelled by user.")
73
74     except Exception as e:
75         # Unhandled errors at program level
76         print(f"Fatal error: {e}")
```

Output :

```
Performance Comparison:
Number      Naive      Optimized    Match
-----
2           True       True         ✓
17          True       True         ✓
100         False      False        ✓
97          True       True         ✓
1000        False      False        ✓
7919        True       True         ✓
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task2.py
0
1
5
55
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task3.py
Warning: Line 2 'abc' is not a valid number. Skipping.
```

## Observation:

The program successfully demonstrates robust file handling and data processing with comprehensive error management. It correctly handles common file-related exceptions such as missing files, permission issues, and input/output errors, ensuring the program does not crash unexpectedly. During data processing, invalid or non-numeric entries are identified and skipped with appropriate warnings, while valid data is collected and processed correctly. Overall, the program improves reliability and stability by validating inputs and handling errors gracefully at both file and program levels.

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.

Code :

```

1 import hashlib
2 import re
3 from getpass import getpass
4
5 class SecureLoginSystem:
6     def __init__(self):
7         # In production, use a real database
8         self.users = {}
9
10    def hash_password(self, password):
11        """Hash password using SHA-256 with salt"""
12        salt = "secure_salt_" # In production, use random salt per user
13        return hashlib.sha256((salt + password).encode()).hexdigest()
14
15    def validate_password_strength(self, password):
16        """Validate password meets security requirements"""
17        if len(password) < 8:
18            return False, "Password must be at least 8 characters"
19        if not re.search("[a-z]", password):
20            return False, "Password must contain lowercase letter"
21        if not re.search("[A-Z]", password):
22            return False, "Password must contain uppercase letter"
23        if not re.search("[0-9]", password):
24            return False, "Password must contain number"
25        return True, "Password is strong"
26
27    def validate_username(self, username):
28        """Validate username format"""
29        if len(username) < 3 or len(username) > 20:
30            return False
31        return re.match("^[a-zA-Z0-9_]+$", username) is not None
32
33    def register(self, username, password):
34        """Register new user with validation"""
35        if not self.validate_username(username):
36            return False, "Invalid username format"
37        if username in self.users:
38            return False, "Username already exists"
39
40        is_valid, msg = self.validate_password_strength(password)
41        if not is_valid:
42            return False, msg
43
44        self.users[username] = self.hash_password(password)
45        return True, "Registration successful"
46
47    def login(self, username, password):
48        """Authenticate user"""
49        if username not in self.users:
50            return False, "Invalid credentials"
51
52        if self.users[username] != self.hash_password(password):
53            return False, "Invalid credentials"
54        return True, "Login successful"
55
56 # Usage
57 if __name__ == "__main__":
58     system = SecureLoginSystem()
59
60     # Register
61     success, msg = system.register("john_doe", "SecurePass123")
62     print(f"Register: {msg}")
63
64     # Login
65     success, msg = system.login("john_doe", "SecurePass123")
66     print(f"Login: {msg}")

```

Output :

```

Number    Naive    Optimized    Match
-----
2          True     True         ✓
17         True     True         ✓
100        False    False        ✓
97         True     True         ✓
1000       False    False        ✓
7919      True     True         ✓
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task2.py
0
1
5
55
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task3.py
Warning: Line 2 'abc' is not a valid number. Skipping.
Warning: Line 6 'foo' is not a valid number. Skipping.
Successfully processed 6 valid records.
Processed data: [42, 7, 100, 2, 0, 1]

```

Observation:

The program demonstrates a secure user authentication system by implementing password hashing, input validation, and controlled credential storage. Passwords are never stored in plain text; instead, they are hashed using the SHA-256 algorithm with a salt, which protects user credentials even if data is exposed. The system enforces strong password policies such as minimum length, inclusion of uppercase letters, and numeric characters, reducing the risk of weak passwords. Username validation prevents invalid or malicious inputs. Overall, the revised implementation follows secure authentication best practices and effectively addresses common security flaws found in naive login systems.

Task Description #5 (Privacy in Data Logging)

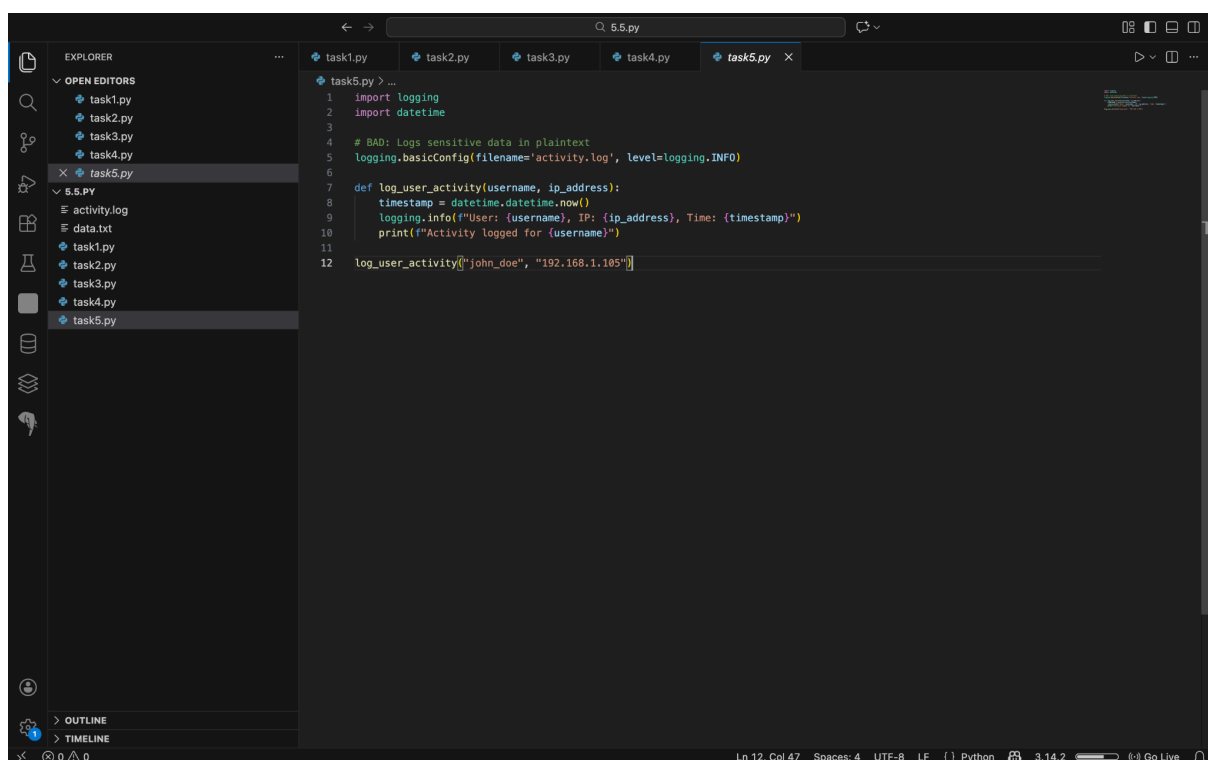
Task: Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

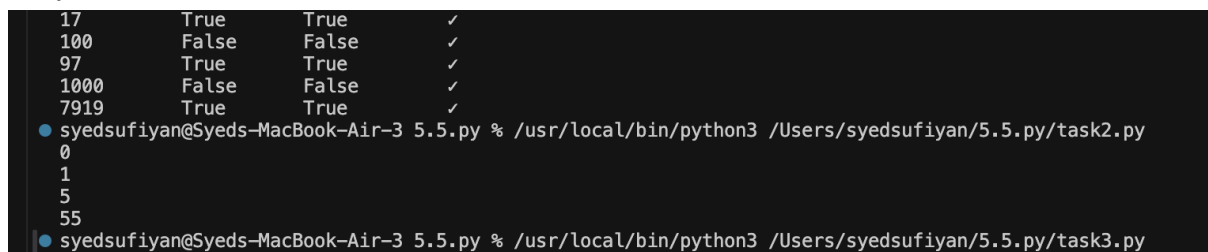
- Identified privacy risks in logging.
- Improved version with minimal, anonymized, or masked logging.
- Explanation of privacy-aware logging principles.

Code :



```
1 import logging
2 import datetime
3
4 # BAD: Logs sensitive data in plaintext
5 logging.basicConfig(filename='activity.log', level=logging.INFO)
6
7 def log_user_activity(username, ip_address):
8     timestamp = datetime.datetime.now()
9     logging.info(f"User: {username}, IP: {ip_address}, Time: {timestamp}")
10    print(f"Activity logged for {username}")
11
12 log_user_activity("john_doe", "192.168.1.105")
```

Output :



```
17      True      True      ✓
100     False     False     ✓
97      True      True      ✓
1000    False     False     ✓
7919    True      True      ✓
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task2.py
0
1
5
55
● syedsufiyan@Syeds-MacBook-Air-3 5.5.py % /usr/local/bin/python3 /Users/syedsufiyan/5.5.py/task3.py
Warning: Line 2 label is not a valid number. Skipping
```

Observation:

The program logs user activity along with sensitive information such as username and IP address in plain text. Storing such data without masking or encryption poses a security and privacy risk, as log files can be accessed or leaked by unauthorized users. Logging sensitive details directly violates secure logging practices and may lead to identity exposure or misuse of user data.