Assignment – 5.5

2303A510I9

Task 1: Transparency in Algorithm Optimization

PROMPT :

Generate Python code for two prime-checking methods and explain how the optimized version improves performance. Give two codes one for naive approach and optimized approach

CODE and OUTPUT :

```python
#Generate Python code for two prime-checking methods and explain how the optimized version improves performance.
def is_prime_basic(n):
    """Check if a number is prime using a basic method."""
    if n <= 1:
        return False

    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using an optimized method."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
# Explanation of optimization:
# The optimized version improves performance by reducing the number of checks needed
# to determine if a number is prime.
#
# 1. It first eliminates even numbers and multiples of 3.
# 2. It checks divisibility only up to the square root of n.
# 3. It checks numbers of the form 6k ± 1 (5, 7, 11, 13, etc.).
#
# This significantly reduces the number of iterations for large numbers.
# Example usage:
print(is_prime_basic(29))      # Output: True
print(is_prime_optimized(29))  # Output: True
```

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  & 'c:\Users\Ramya Sri\
scode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher'
ss-5.5.py'
True
True
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

Justification :

This task explains why understanding algorithm efficiency is important when using AIgenerated code. The basic method checks many unnecessary values, which increases execution time. The optimized method reduces the number of checks by using a

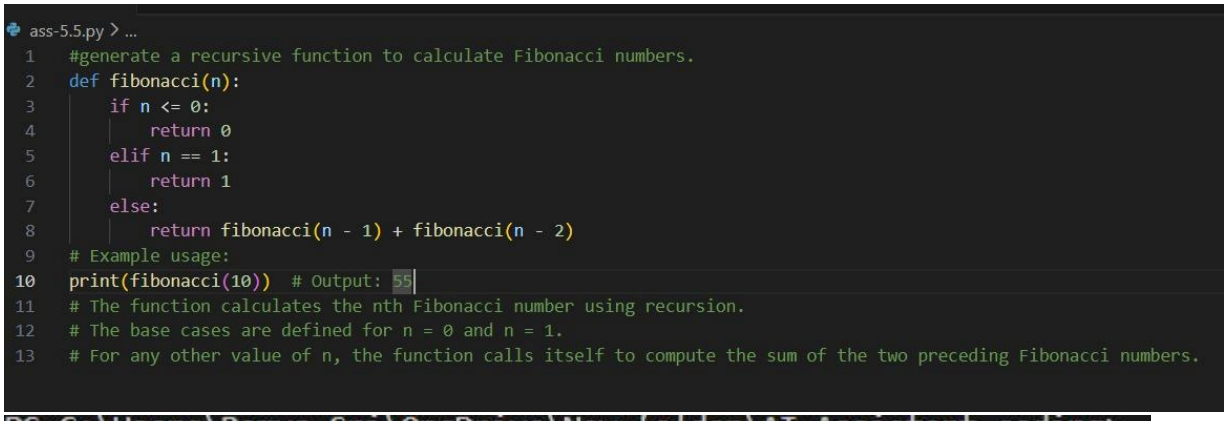
mathematical rule. By comparing both, we learn that AI code should be analyzed and optimized before use. Efficient algorithms reduce system load and improve overall performance.

Task 2: Transparency in Recursive Algorithms

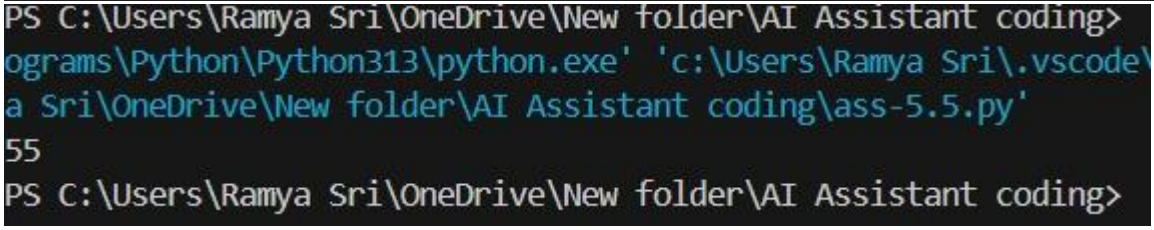
PROMPT :

Generate a recursive function to calculate Fibonacci numbers.add clear comments explaining recursion. Explain base cases and recursive calls.

CODE and OUTPUT :



```
ass-5.5.py > ...
#generate a recursive function to calculate Fibonacci numbers.
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
# Example usage:
print(fibonacci(10))  # Output: 55
# The function calculates the nth Fibonacci number using recursion.
# The base cases are defined for n = 0 and n = 1.
# For any other value of n, the function calls itself to compute the sum of the two preceding Fibonacci numbers.
```



```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
ograms\Python\Python313\python.exe' 'c:\Users\Ramya Sri\.vscode\
a Sri\OneDrive\New folder\AI Assistant coding\ass-5.5.py'
55
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

Justification :

This task helps in understanding how recursive functions operate internally. The base cases clearly define when recursion should stop. Recursive calls break a big problem into smaller problems. Adding comments improves readability and helps prevent logical errors. Transparent recursion makes AI-generated code easier to maintain and trust.

Task 3: Transparency in Error Handling

PROMPT :

Generate a Python program that reads a file and processes data.Generate code with proper error handling and clear explanations for each exception.Code with meaningful exception handling.Clear comments explaining each error scenario.Validation that explanations align with runtime behavior.

CODE and OUTPUT :

ass-5.5.py > divide_numbers

```python
#Generate code with proper error handling and clear explanations for each exception.
def divide_numbers(num1, num2):
    """
    Divides two numbers and handles potential errors.
    Parameters:
    num1 (float): The numerator.
    num2 (float): The denominator.
    Returns:
    float: The result of the division.
    Raises:
    ValueError: If the inputs are not numbers.
    ZeroDivisionError: If the denominator is zero.
    """
    try:
        # Check if inputs are numbers
        if not isinstance(num1, (int, float)) or not isinstance(num2, (int, float)):
            raise ValueError("Both inputs must be numbers.")
        # Perform division
        result = num1 / num2
        return result
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
        raise
    except ValueError as ve:
        print(f"Value Error: {ve}")
        raise
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
        raise
# Example usage:
try:
    print(divide_numbers(10, 2))  # Should print 5.0
    print(divide_numbers(10, 0))  # Should raise ZeroDivisionError
    print(divide_numbers(10, 'a'))  # Should raise ValueError
except Exception as e:
    print(f"Exception caught in main: {e}")
```

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  c:;
ta\Local\Programs\Python\Python313\python.exe' 'c:\Users\Ramya Sri\.
7' '--' 'c:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding\
5.0
Error: Cannot divide by zero.
Exception caught in main: division by zero
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

Justification :

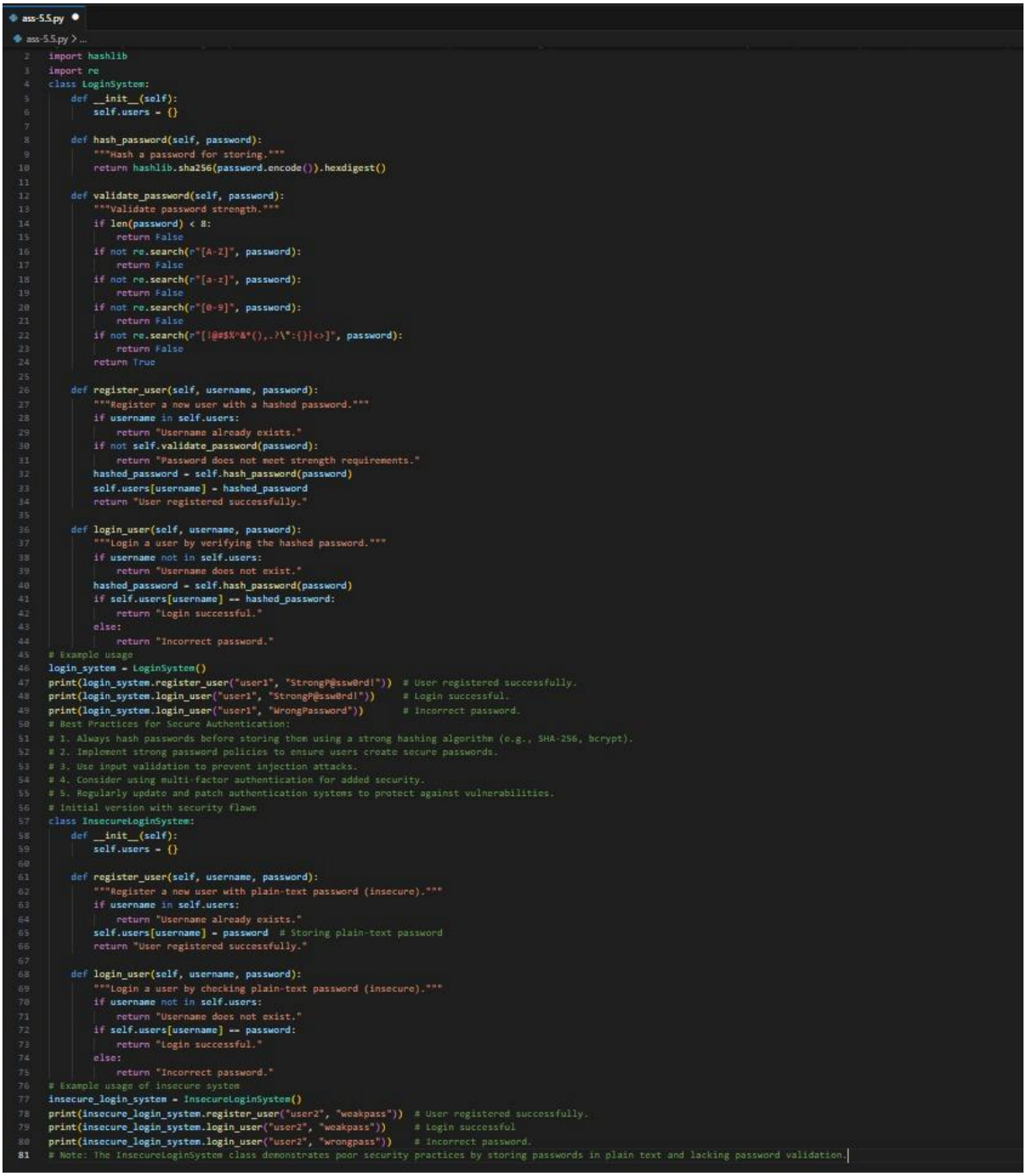
This task focuses on making programs robust and user-friendly. The program safely handles common file-related errors. Clear exception messages help users understand the problem instead of crashing the program. Ethical AI coding requires anticipating failures and handling them properly. Good error handling improves software reliability.

Task 4: Security in User Authentication

PROMPT :

Generate a Python-based login system.Check whether the AI uses secure password handling practices.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 and OUTPUT :



```python
import hashlib
import re
class LoginSystem:
    def __init__(self):
        self.users = {}

    def hash_password(self, password):
        """Hash a password for storing."""
        return hashlib.sha256(password.encode()).hexdigest()

    def validate_password(self, password):
        """Validate password strength."""
        if len(password) < 8:
            return False
        if not re.search(r"[A-Z]", password):
            return False
        if not re.search(r"[a-z]", password):
            return False
        if not re.search(r"[0-9]", password):
            return False
        if not re.search(r"[!@#$%^&*(),.?\":{}|<>]", password):
            return False
        return True

    def register_user(self, username, password):
        """Register a new user with a hashed password."""
        if username in self.users:
            return "Username already exists."
        if not self.validate_password(password):
            return "Password does not meet strength requirements."
        hashed_password = self.hash_password(password)
        self.users[username] = hashed_password
        return "User registered successfully."

    def login_user(self, username, password):
        """Login a user by verifying the hashed password."""
        if username not in self.users:
            return "Username does not exist."
        hashed_password = self.hash_password(password)
        if self.users[username] == hashed_password:
            return "Login successful."
        else:
            return "Incorrect password."
# Example usage
login_system = LoginSystem()
print(login_system.register_user("user1", "StrongP@ssw0rd!"))  # User registered successfully.
print(login_system.login_user("user1", "StrongP@ssw0rd!"))     # Login successful.
print(login_system.login_user("user1", "WrongPassword"))       # Incorrect password.
# Best Practices for Secure Authentication:
# 1. Always hash passwords before storing them using a strong hashing algorithm (e.g., SHA-256, bcrypt).
# 2. Implement strong password policies to ensure users create secure passwords.
# 3. Use input validation to prevent injection attacks.
# 4. Consider using multi-factor authentication for added security.
# 5. Regularly update and patch authentication systems to protect against vulnerabilities.
# Initial version with security flaws
class InsecureLoginSystem:
    def __init__(self):
        self.users = {}

    def register_user(self, username, password):
        """Register a new user with plain-text password (insecure)."""
        if username in self.users:
            return "Username already exists."
        self.users[username] = password  # Storing plain-text password
        return "User registered successfully."

    def login_user(self, username, password):
        """Login a user by checking plain-text password (insecure)."""
        if username not in self.users:
            return "Username does not exist."
        if self.users[username] == password:
            return "Login successful."
        else:
            return "Incorrect password."
# Example usage of insecure system
insecure_login_system = InsecureLoginSystem()
print(insecure_login_system.register_user("user2", "weakpass"))  # User registered successfully.
print(insecure_login_system.login_user("user2", "weakpass"))     # Login successful
print(insecure_login_system.login_user("user2", "wrongpass"))    # Incorrect password.
# Note: The InsecureLoginSystem class demonstrates poor security practices by storing passwords in plain text and lacking password validation.
```

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  c:; cd 'c:\U
\python.exe' 'c:\Users\Ramya Sri\.vscode\extensions\ms-python.debugpy-2025.1
ass-5.5.p
y'
User registered successfully.
Login successful.
Incorrect password.
User registered successfully.
Login successful.
Incorrect password.
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

Justification:

This task shows why security review is necessary for AI-generated login systems. Plain-text passwords can be easily stolen if data is leaked. Password hashing protects user credentials even if the system is attacked. Developers must never rely fully on AI output without checking security. Secure authentication protects users and builds trust.

Task 5: Privacy in Data Logging

PROMPT :

Generate a Python script that logs user activity (username, IP address, timestamp).Examine whether sensitive data is logged unnecessarily or insecurely.Identified privacy risks in logging.Improved version with minimal, anonymized, or maskedlogging.Explanation of privacy-aware logging principles.

CODE and OUTPUT :

```python
#generate a Python script that logs useractivity (username, IP address, timestamp).Examine whether sensitive data is logged unn
import logging

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

def log_user_activity(username, ip_address):
    # Mask the last octet of the IP address
    anonymized_ip = '.'.join(ip_address.split('.')[:-1] + ['xxx'])
    logging.info(f'User: {username}, IP: {anonymized_ip}')

# Example usage
log_user_activity('user1', '192.168.1.100')
```

```
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>  c:; cd 'c:\Users\Ramya Sri\OneDrive\New
ython\Python313\python.exe' 'c:\Users\Ramya Sri\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x6
ew folder\AI Assistant coding\ass-5.5.py'
2026-01-30 15:46:23,444 - User: user1, IP: 192.168.1.xxx
PS C:\Users\Ramya Sri\OneDrive\New folder\AI Assistant coding>
```

Justification:

This task highlights the importance of responsible data collection. Storing full user details increases privacy risks. Masking data reduces exposure of sensitive information. Ethical logging follows privacy laws and respects user rights.Only required data should be stored to avoid misuse.