

AI ASSISTED CODING

ASSIGNMENT-5.5

NAME:D.PAVAN

ROLL NO: 2403A52L07

BATCH-50

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.**

PROMPT:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

OUTPUT:

The screenshot shows a Visual Studio Code (VS Code) interface. The Explorer sidebar on the left lists a project structure with folders like 'week3', 'week3.1', 'week3.2', and 'week3.3'. Inside 'week3.3' are several Python files: 'Task1.py', 'Task2.py', 'Task3.py', 'Task4.py', 'Task5.py', 'Task6.py', 'Task7.py', 'Task8.py', and 'Task9.py'. The 'Task1.py' file is open in the main editor area. It contains two functions: 'is_prime_basic(n)' and 'is_prime_optimized(n)'. The code is annotated with AI-generated comments explaining the purpose of each function and how it improves performance.

```
# Generated 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
```

The terminal at the bottom shows the output of running the code with the number 5 as input. Both the basic and optimized methods correctly identify 5 as a prime number.

EXPLANATION:

The naive prime-checking method tests divisibility from 2 up to $n-1$, which is inefficient with time complexity $O(n)$. The optimized method reduces checks by stopping at \sqrt{n} and skipping even numbers, improving efficiency to $O(\sqrt{n})$. This comparison highlights how algorithmic optimization saves computation time.

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.

PROMPT:

“Generate a recursive Python function for Fibonacci numbers with clear comments explaining base cases and recursive calls.”

OUTPUT:

```

File: Task2.py
-----
1 # Generate a recursive Python function for Fibonacci numbers with clear
2 # comments explaining base cases and recursive calls...
3
4 def fibonacci(n):
5     """Return the nth Fibonacci number using recursion."""
6
7     # Base case: the first Fibonacci number is 0
8     if n == 0:
9         return 0
10    # Base case: the second Fibonacci number is 1
11    elif n == 1:
12        return 1
13    else:
14        # Recursive case: sum of the two preceding Fibonacci numbers
15        return fibonacci(n - 1) + fibonacci(n - 2)
16
17 # Take user input for which Fibonacci Number to compute
18 num = int(input("Enter a non-negative integer to find the Fibonacci number: "))
19
20 if num < 0:
21     print("Please enter a non-negative integer!")
22 else:
23     result = fibonacci(num)
24     print(f"The {num}th Fibonacci number is: {result}")

```

EXPLANATION:

Recursion works by breaking a problem into smaller subproblems until reaching base cases. The base cases $f(0)=0$ and $f(1)=1$ stop infinite calls, while recursive calls build the solution step by step. Clear comments make the recursive flow transparent and easier to trace.

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.**

PROMPT:

“Generate code with proper error handling and clear explanations for each exception. Generate Python program that reads a file and processes data.”

OUTPUT:

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a project structure with files like `Task1.py`, `Task2.py`, `Task3.py`, `Task4.py`, `Task5.py`, and `Task6.py` under the `week3` folder.
- Code Editor:** The `Task3.py` file is open, displaying Python code for reading and processing a file with error handling. The code includes comments explaining the purpose of the program and how it handles exceptions for file not found and permission errors.
- Output Panel:** Shows the command `thon.exe c:/Users/krish/Downloads/notes.txt` and the message "Enter the path to the file you want to read: C:/Users/krish/Downloads/notes.txt". It also shows "The file contains 0 words."
- Sidebar:** On the right, there's a sidebar titled "Build with Agent" which includes a message about AI responses being inaccurate, instructions to enable the Agent, and a "Show More" button.
- Bottom Bar:** Includes icons for file operations, a search bar, and tabs for "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "ISSUES", and "PORTS".

EXPLANATION:

The program uses try-except blocks to handle errors such as missing files, permission issues, or invalid paths. Each exception is explained clearly, ensuring the program does not crash unexpectedly. Transparent error handling improves reliability and user experience.

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

PROMPT:

“Generate a Python-based login system and explain whether it uses secure password handling.”

OUTPUT:

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a file tree with a folder named "week5" containing subfolders "week3", "week4", and "week5". The "week5" folder contains files "task1.py", "task2.py", "task3.py", and "task4.py".
- Code Editor:** Displays the content of "task4.py". The code is a Python script for a login system. It includes imports for hashlib, defines a hash_password function, and a validate_input function. It then defines a login_system function that prompts the user for a username and password, validates them, and prints a success message if valid.
- Terminal:** At the bottom, the terminal window shows the output of running the script: "Enter your username: user1", "Enter your password: securpassword", and "Login successful!".
- Status Bar:** Shows the current file as "task4.py", the line number as "1", the column as "14", the workspace as "Scenes A", the encoding as "UTF-8", the Python version as "Python 3.11 (64-bit)", and the file type as "Python".

EXPLANATION:

Plain-text password storage is insecure because it exposes sensitive data. The improved version uses hashing and input validation, which protect user credentials. Secure authentication practices emphasize the importance of safeguarding user data and preventing 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.

PROMPT:

“Generate a Python script that logs user activity (username, IP, timestamp).”

Analyze privacy risks and revise to use minimal, anonymized logging."

OUTPUT:

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of files. The `week5` folder contains `Task1.py`, `Task2.py`, `Task3.py`, `Task4.py`, `Task5.py`, and `Task6.py`. The `week5.5` folder contains `Task1.py`, `Task2.py`, `Task3.py`, `Task4.py`, `Task5.py`, and `user-activity.log`.
- Code Editor:** The `Task5.py` file is open. It contains Python code to log user activity with anonymized IP addresses. The code includes imports for `logging` and `datetime`, and defines a function `anonymize_ip` that masks the last octet of an IP address.
- Terminal:** The terminal shows the command `python.exe c:/Users/Krish/OneDrive/Desktop/AIAC/week3.5/week5.5.py/Task5.py` being run, followed by a warning about the deprecation of `datetime.datetime.utctimetuple()`. The output also shows the timestamp converted to UTC.
- Status Bar:** The status bar at the bottom indicates the file is a Python file (`Python`), the line number is 17, the column is 100, the encoding is UTF-8, and the file size is 1.1 MB.

EXPLANTION:

Logging sensitive data such as usernames or IP addresses can create privacy risks. The improved version uses anonymization or minimal logging to reduce

exposure. Privacy-aware coding ensures only necessary information is collected, protecting user identity and maintaining ethical standards.