

AI ASSISTANT CODING

ASSIGNMENT-5.5

Name: B. Akhira Nandhini

Hallticket:2303A51516

Batch:22

Task Description #1 (Transparency in Algorithm Optimization)

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

- Naive approach(basic)
- Optimized approach

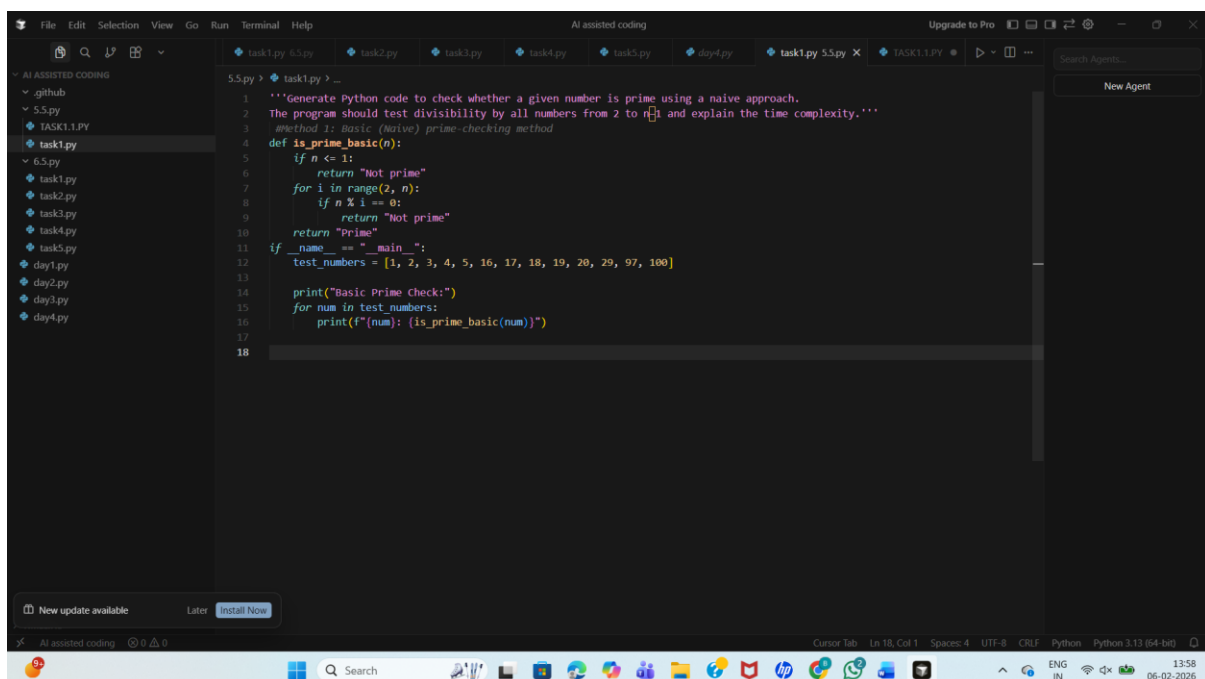
Prompt:

“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:



The screenshot shows a code editor with a dark theme. The left sidebar displays a file explorer with a tree structure under 'AI ASSISTED CODING'. The main editor area shows a Python file named 'task1.py' with the following code:

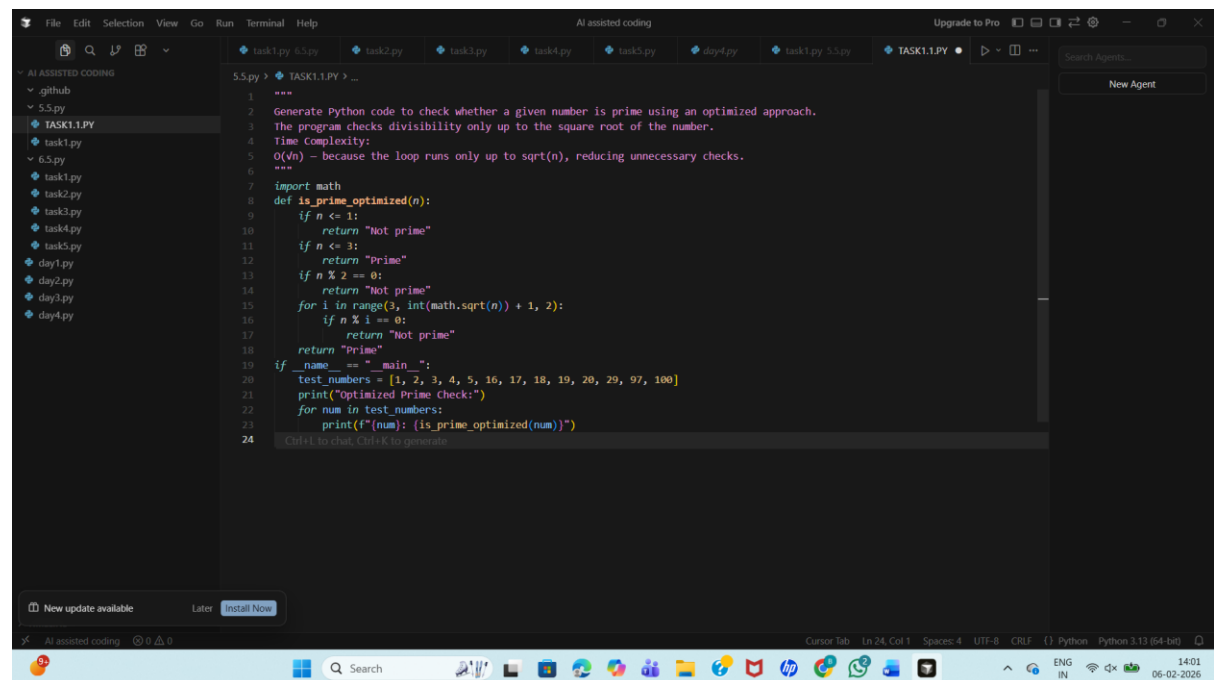
```
1 '''Generate Python code to check whether a given number is prime using a naive approach.
2 The program should test divisibility by all numbers from 2 to n-1 and explain the time complexity.'''
3 #method 1: Basic (naive) prime-checking method
4 def is_prime_basic(n):
5     if n <= 1:
6         return "Not prime"
7     for i in range(2, n):
8         if n % i == 0:
9             return "Not prime"
10    return "Prime"
11
12 if __name__ == "__main__":
13     test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
14
15     print("Basic Prime Check:")
16     for num in test_numbers:
17         print(f"{num}: {is_prime_basic(num)}")
18
```

The status bar at the bottom indicates the cursor is at line 18, column 1, with 4 spaces, in UTF-8 encoding, using the CRLF line ending, and the Python 3.13 (64-bit) interpreter is selected.

Output:

```
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\Users\abhin\AppData\Local\Programs\Python\Python313\python.exe
eDrive/Desktop/AI assisted coding/5.5.py/task1.py"
Basic Prime Check:
1: Not prime
2: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
17: Prime
18: Not prime
19: Prime
20: Not prime
29: Prime
97: Prime
100: Not prime
```

METHOD 2:

A screenshot of a code editor window titled "AI assisted coding". The editor shows a Python script named "TASK1.1.PY" with the following content:

```
1 """
2 Generate Python code to check whether a given number is prime using an optimized approach.
3 The program checks divisibility only up to the square root of the number.
4 Time Complexity:
5 O(sqrt(n)) - because the loop runs only up to sqrt(n), reducing unnecessary checks.
6 """
7 import math
8 def is_prime_optimized(n):
9     if n <= 1:
10         return "Not prime"
11     if n <= 3:
12         return "Prime"
13     if n % 2 == 0:
14         return "Not prime"
15     for i in range(3, int(math.sqrt(n)) + 1, 2):
16         if n % i == 0:
17             return "Not prime"
18     return "Prime"
19 if __name__ == "__main__":
20     test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
21     print("Optimized Prime Check:")
22     for num in test_numbers:
23         print(f"{num}: {is_prime_optimized(num)}")
24
```

The editor interface includes a sidebar on the left with a file explorer showing a project structure with folders like ".github" and files like "5.5.py", "task1.py", "task2.py", etc. The bottom status bar indicates the current file is "TASK1.1.PY" in "Python" mode, with a cursor at line 24, column 1. The system tray at the bottom shows the date and time as "1401 06-02-2026".

Output:

Optimized Prime Check:

1: Not prime

2: Prime

3: Prime

4: Not prime

5: Prime

16: Not prime

17: Prime

18: Not prime

19: Prime

20: Not prime

29: Prime

97: Prime

100: Not prime

PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> █

Explanation:

- In the naive approach, the program checks whether a number is prime by testing its divisibility with **all integers from 2 to $n-1$** .
If the number is divisible by any of these values, it is not a prime number; otherwise, it is considered prime.
- This method is easy to understand and implement, but it performs a large number of unnecessary checks, especially for bigger numbers.
- In the optimized approach, the program checks divisibility **only up to the square root of the number**.
This is based on the mathematical fact that if a number has a factor greater than \sqrt{n} , it must also have a corresponding factor smaller than \sqrt{n} .
- Additionally, even numbers are skipped after checking divisibility by 2, which further reduces the number of iterations

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  # generate a python code for recursive function to calculate fibonacci numbers
2  # add clear comments explaining the approach used in the code
3  # explain base cases and recursive calls
4  # =====
5  # 1. RECURSIVE FIBONACCI FUNCTION
6  def fibonacci_recursive(n):
7      # Base case: if n is 0, return 0
8      if n == 0:
9          return 0
10     # Base case: if n is 1, return 1
11     elif n == 1:
12         return 1
13     # Recursive case: calculate fibonacci(n-1) + fibonacci(n-2)
14     else:
15         return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
16
17 # 2. TESTING & VERIFICATION
18 if __name__ == "__main__":
19     print("-" * 60)
20     print("FIBONACCI NUMBERS USING RECURSION")
21     print("-" * 60)
22
23     test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
24
25     for n in test_cases:
26         print(f"Fibonacci({n}) = {fibonacci_recursive(n)}")
27

```

Output:

```

PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\Users\abhin\OneDrive\Desktop\AI assisted coding\5.5.py/task2.py"
=====
FIBONACCI NUMBERS USING RECURSION
=====
Fibonacci(0) = 0
Fibonacci(1) = 1
Fibonacci(2) = 1
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>

```

Explanation:

The Fibonacci series is a sequence of numbers in which each number is the sum of the previous two numbers.

The sequence starts as: **0, 1, 1, 2, 3, 5, 8, ...**

This program uses **recursion**, where a function calls itself to solve a problem by breaking it into smaller subproblems.

Task Description #3 (Transparency in Error Handling)

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

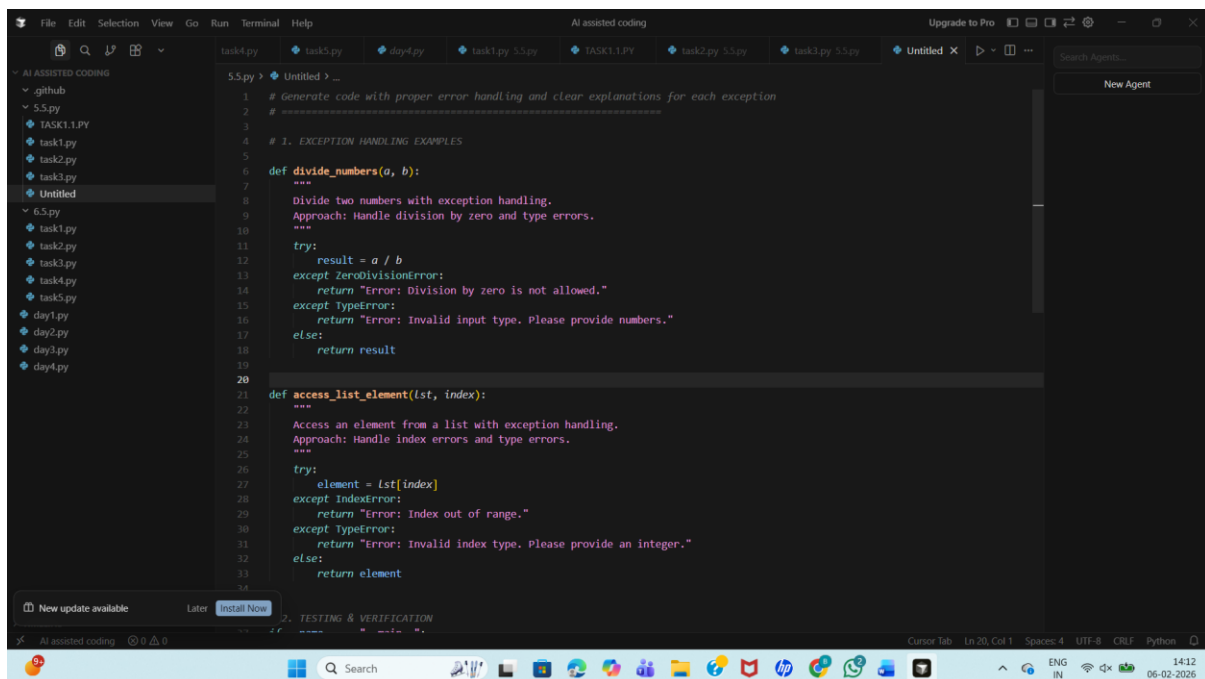
Prompt:

“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.

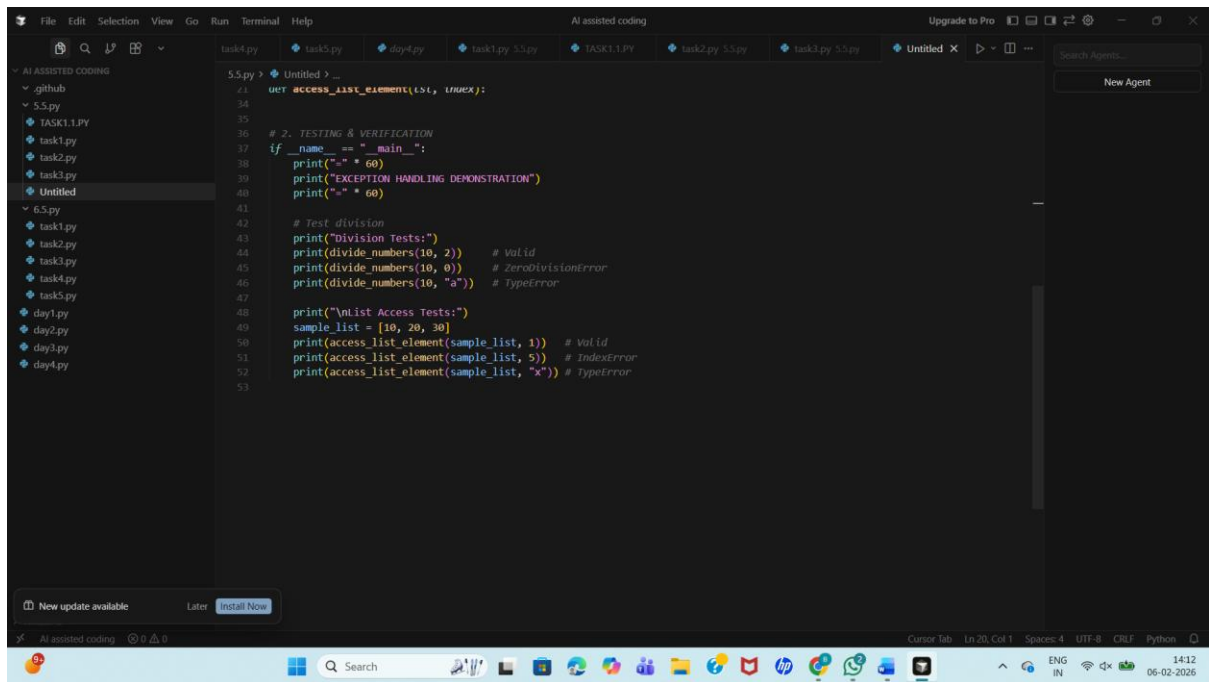
code:



The screenshot shows a code editor with a dark theme. The left sidebar displays a file explorer with a tree view containing files like .github, 5.5.py, TASK1.1.PY, task1.py, task2.py, task3.py, and an 'Untitled' file. The main editor area shows the content of '5.5.py'. The code is a Python script with two functions: 'divide_numbers(a, b)' and 'access_list_element(lst, index)'. Both functions include docstrings and use try-except blocks to handle ZeroDivisionError, IndexError, and TypeError. The code is as follows:

```
1 # Generate code with proper error handling and clear explanations for each exception
2 # =====
3
4 # 1. EXCEPTION HANDLING EXAMPLES
5
6 def divide_numbers(a, b):
7     """
8     Divide two numbers with exception handling.
9     Approach: Handle division by zero and type errors.
10    """
11    try:
12        result = a / b
13    except ZeroDivisionError:
14        return "Error: Division by zero is not allowed."
15    except TypeError:
16        return "Error: Invalid input type. Please provide numbers."
17    else:
18        return result
19
20
21 def access_list_element(lst, index):
22     """
23     Access an element from a list with exception handling.
24     Approach: Handle index errors and type errors.
25    """
26    try:
27        element = lst[index]
28    except IndexError:
29        return "Error: Index out of range."
30    except TypeError:
31        return "Error: Invalid index type. Please provide an integer."
32    else:
33        return element
34
```

The bottom status bar of the editor shows 'Cursor tab', 'Ln 20, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', and 'Python'. The Windows taskbar at the very bottom shows the search bar, task view, and several application icons, with the system clock displaying '14:12' and '06-02-2026'.



Output:

```
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding> & C:\Users\abhin\OneDrive\Desktop\AI assisted coding\5.5.py\Untitled"
=====
EXCEPTION HANDLING DEMONSTRATION
=====
Division Tests:
5.0
Error: Division by zero is not allowed.
Error: Invalid input type. Please provide numbers.

List Access Tests:
20
Error: Index out of range.
Error: Invalid index type. Please provide an integer.
PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>
```

Explanation:

Exception handling is a mechanism in Python used to handle runtime errors so that the program does not crash and can continue executing smoothly. It helps in writing reliable and user-friendly programs.

Task Description #4 (Security in User Authentication)

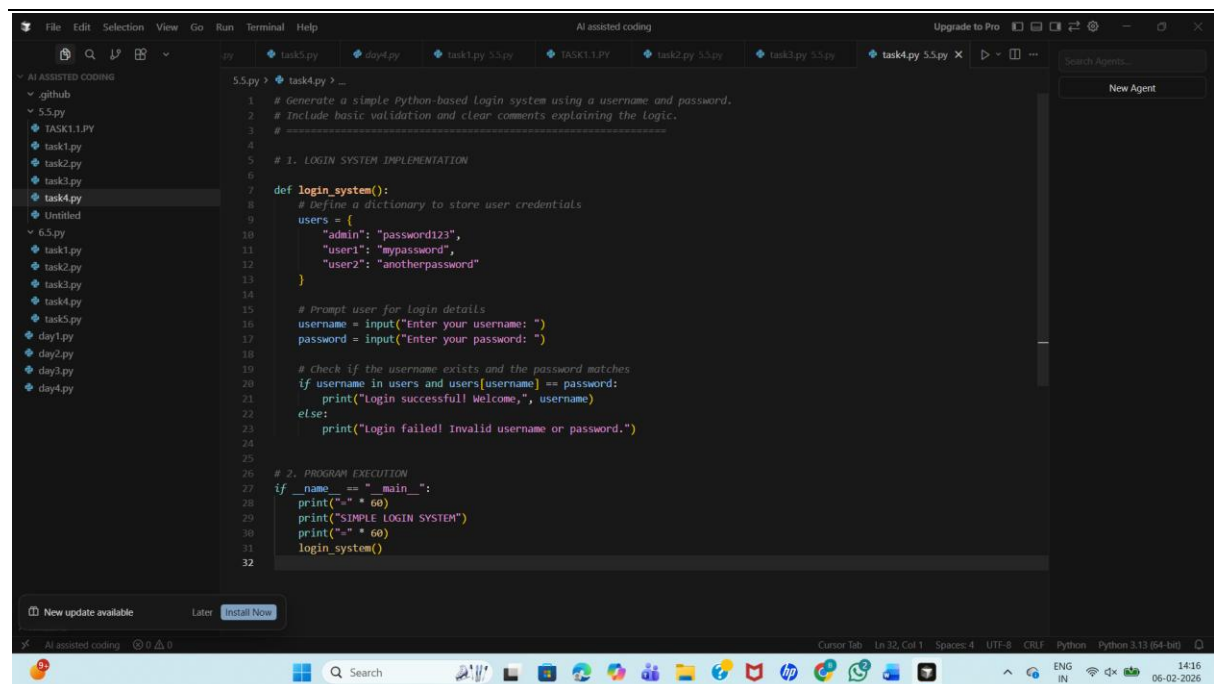
Task: Use an AI tool to generate a Python-based login system.

Analyze: Check whether the AI uses secure password handling practices.

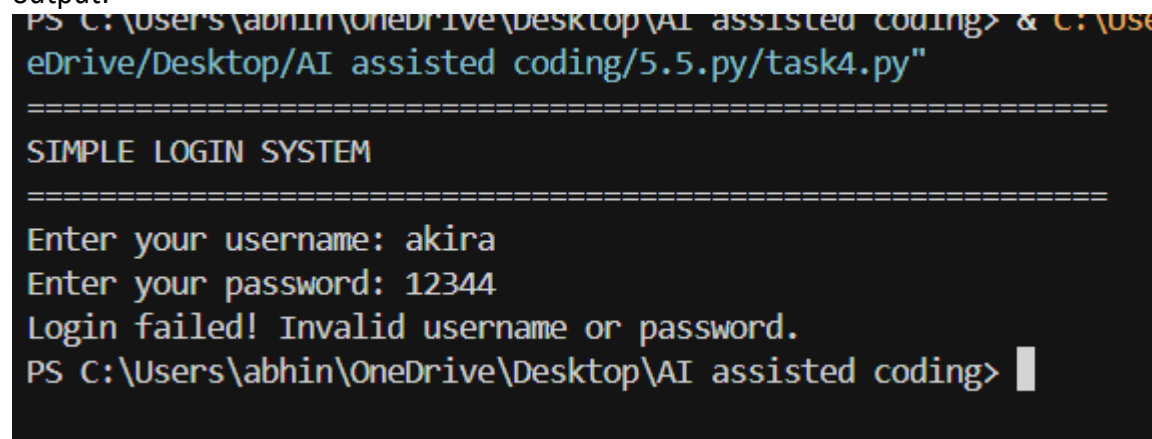
Expected Output:

- 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.

Code:

A screenshot of a code editor interface. The left sidebar shows a file explorer with a project named 'AI ASSISTED CODING' containing files like '5.5.py', 'task1.py', 'task2.py', 'task3.py', 'task4.py', 'task5.py', 'day1.py', 'day2.py', 'day3.py', and 'day4.py'. The main editor area displays the code for 'task4.py'. The code is a Python script for a simple login system. It includes comments at the top: '# Generate a simple python-based login system using a username and password.' and '# Include basic validation and clear comments explaining the logic.' The script is divided into two sections: '# 1. LOGIN SYSTEM IMPLEMENTATION' and '# 2. PROGRAM EXECUTION'. The first section defines a 'login_system()' function that uses a dictionary to store user credentials (admin, user1, user2) and prompts the user for a username and password. It then checks if the entered credentials match the stored ones. The second section contains a main block that prints a title and calls the login_system function. The bottom status bar shows 'Cursor Tab', 'Ln 32, Col 1', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and 'Python 3.11 (64-bit)'.

output:

A screenshot of a terminal window. The prompt is 'PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>'. The user has entered the command '& C:\Users\abhin\OneDrive\Desktop\AI assisted coding\5.5.py\task4.py'. The output shows a title 'SIMPLE LOGIN SYSTEM' followed by prompts for 'Enter your username: akira' and 'Enter your password: 12344'. The response is 'Login failed! Invalid username or password.' The prompt returns to 'PS C:\Users\abhin\OneDrive\Desktop\AI assisted coding>'. The terminal background is dark with light-colored text.

Explanation: This program implements a basic login system using Python. It verifies a user by checking the entered username and password against stored credentials.

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 :

```
logactivity.py >--
1 # "Generate a Python script that logs user activity including username, IP address, and t
2 # =====
3 import logging
4 from datetime import datetime
5 # 1. LOGGING SETUP
6 # Configure logging to write to a file with the specified format
7 logging.basicConfig(
8     filename='user_activity.log',
9     level=logging.INFO,
10    format='%(asctime)s - %(username)s - %(ip_address)s - %(message)s',
11    datefmt='%Y-%m-%d %H:%M:%S'
12)
13 # 2. FUNCTION TO LOG USER ACTIVITY
14 def log_user_activity(username, ip_address):
15     """
16     Log user activity with username, IP address, and timestamp.
17     Approach: Use the logging module to log the information.
18     """
19     logging.info('User logged in', extra={'username': username, 'ip_address': ip_address})
20 # example usage
21 if __name__ == "__main__":
22     print("-" * 60)
23     print("USER ACTIVITY LOGGING")
24     print("-" * 60)
25
26     # Sample user activity logging
27     users = [
28         ("alice", "192.168.1.100"),
29         ("bob", "192.168.1.101"),
30     ]
31
32     for
33         print(f"Logged activity for user: {username}, IP: {ip_address}")
```

OUTPUT :

```
===== ...
(.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/logactivi
ty.py
=====
USER ACTIVITY LOGGING
=====
Logged activity for user: alice, IP: 192.168.1.100
Logged activity for user: bob, IP: 192.168.1.101
(.venv) PS D:\AIASSCoding> 
```

FINAL DESCRIPTION :

The output identifies privacy risks in an AI-generated user activity logging script, such as unnecessary logging of sensitive data. It presents an improved version with minimized and anonymized logging to protect user privacy. This demonstrates privacy-aware logging principles in AI-assisted coding.