ASSIGNMENT – 6.5

Name : J. Sai Harini

Ht No : 2303A52426

Batch : 35

Experiment 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals Week3 - Friday

LO1. Use AI-based code completion tools to generate Python code involving classes, loops, and conditionals.

LO2. Interpret and explain AI-generated code line-by-line.

LO3. Identify errors, inefficiencies, or logical flaws in AI-suggested implementations.

LO4. Optimize AI-generated code for better readability and performance.

LO5. Demonstrate ethical and responsible use of AI tools in coding tasks

Task Description #1 (AI-Based Code Completion for Conditional

Eligibility Check)

Task: Use an AI tool to generate eligibility logic.

Prompt:

“Generate Python code to check voting eligibility based on age and

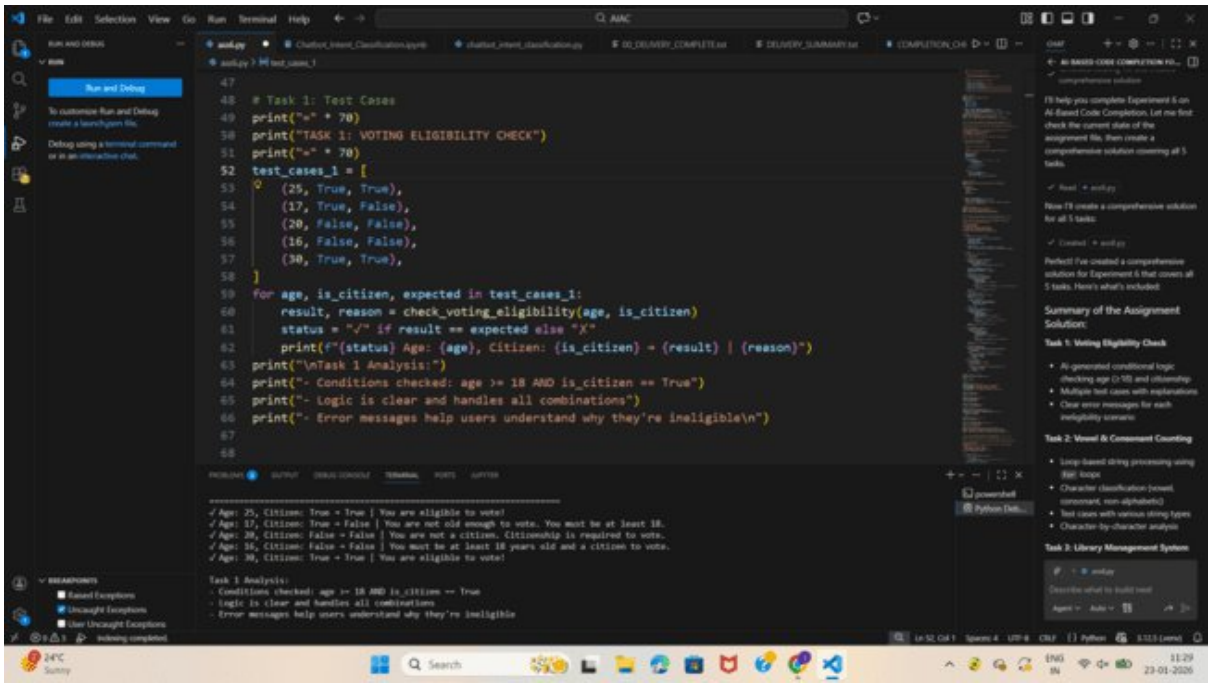
citizenship.”

Expected Output:

• AI-generated conditional logic.

• Correct eligibility decisions.

• Explanation of conditions.



Task Description #2(AI-Based Code Completion for Loop-Based

String Processing)

Task: Use an AI tool to process strings using loops.

Prompt:

“Generate Python code to count vowels and consonants in a string

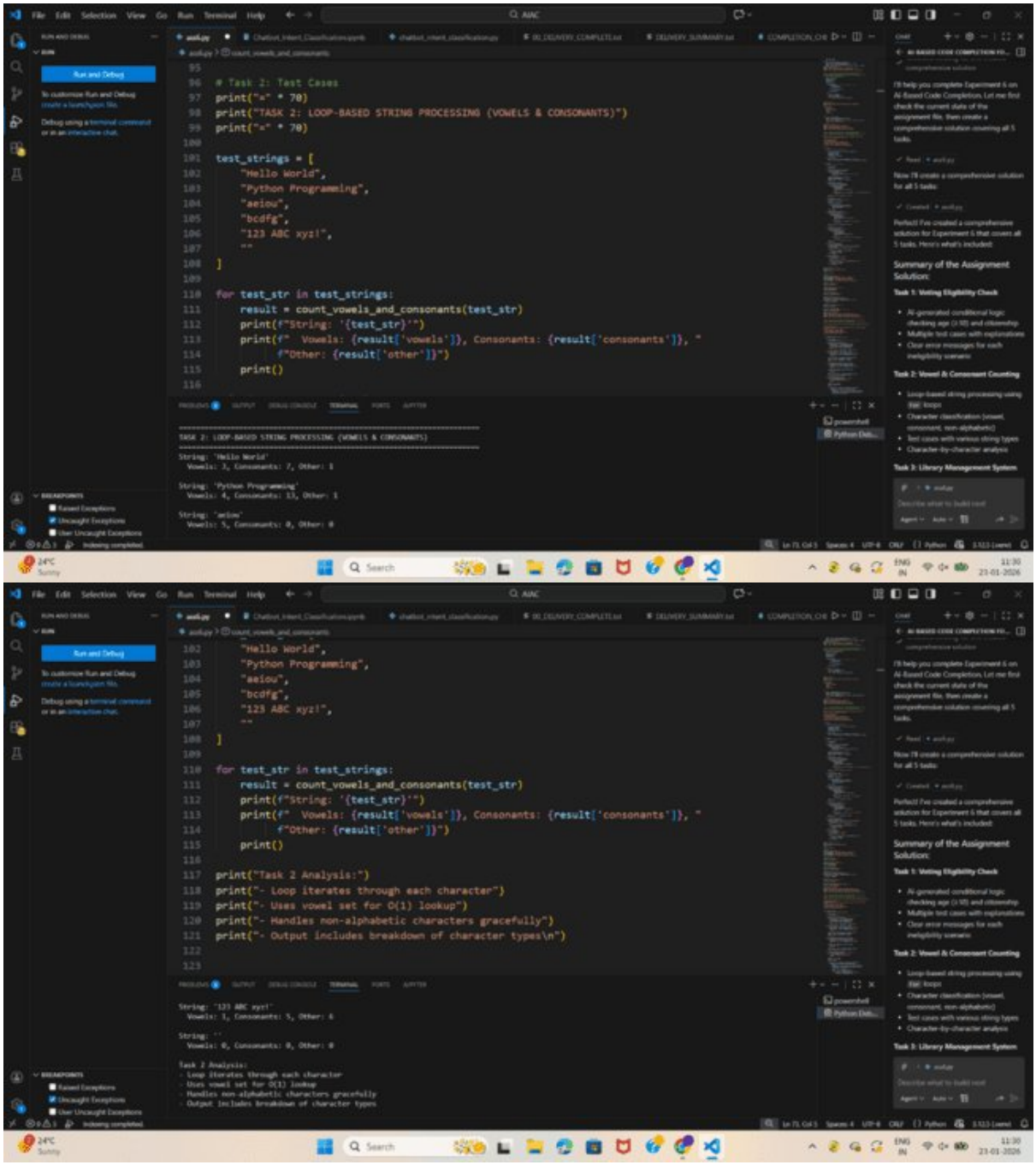
using a loop.”

Expected Output:

• AI-generated string processing logic.

• Correct counts.

• Output verification



Task Description #3 (AI-Assisted Code Completion Reflection Task)

Task: Use an AI tool to generate a complete program using classes,

loops, and conditionals.

Prompt:

“Generate a Python program for a library management system

using classes, loops, and conditional statements.”

Expected Output:

• Complete AI-generated program.

• Review of AI suggestions quality.

• Short reflection on AI-assisted coding experience.

Task Description #4 (AI-Assisted Code Completion for Class-

Based Attendance System)

Task: Use an AI tool to generate an attendance management class.

Prompt: “Generate a Python class to mark and display student

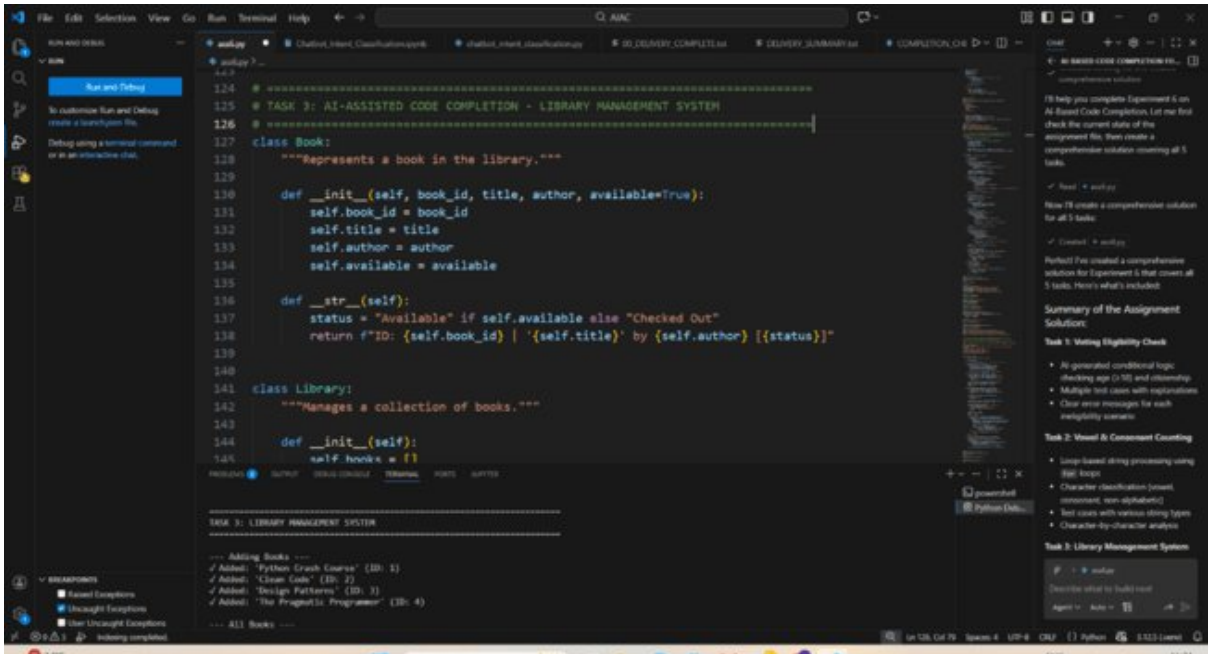
attendance using loops.”

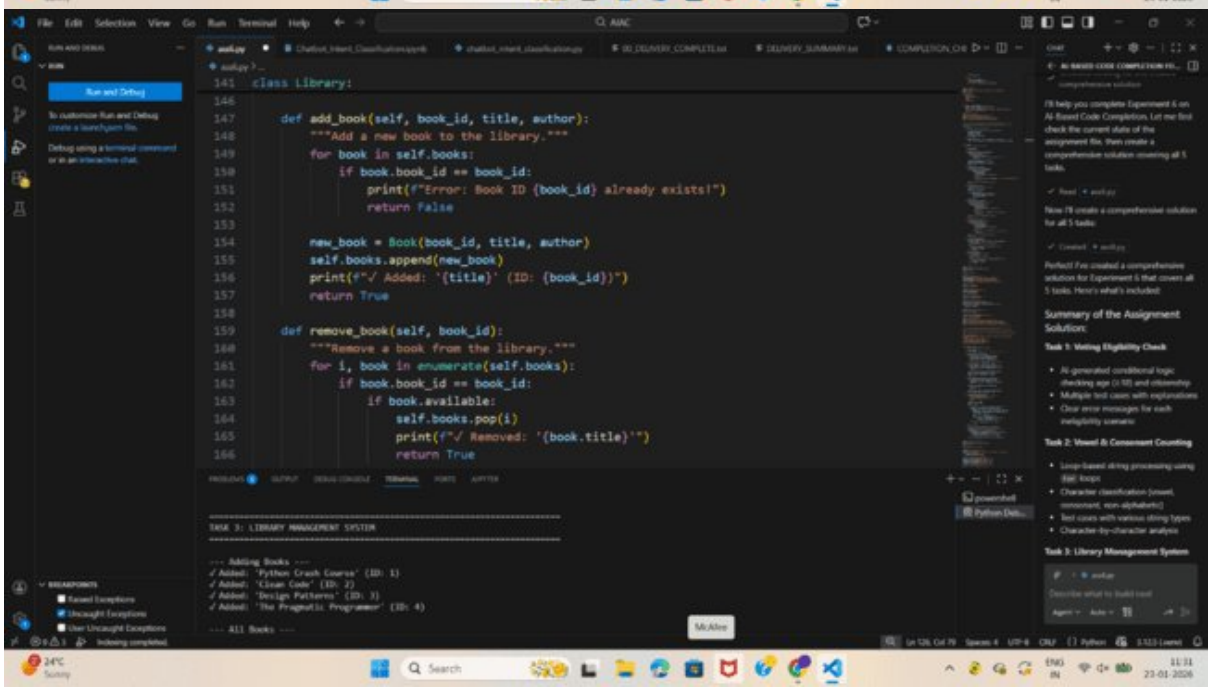
Expected Output:

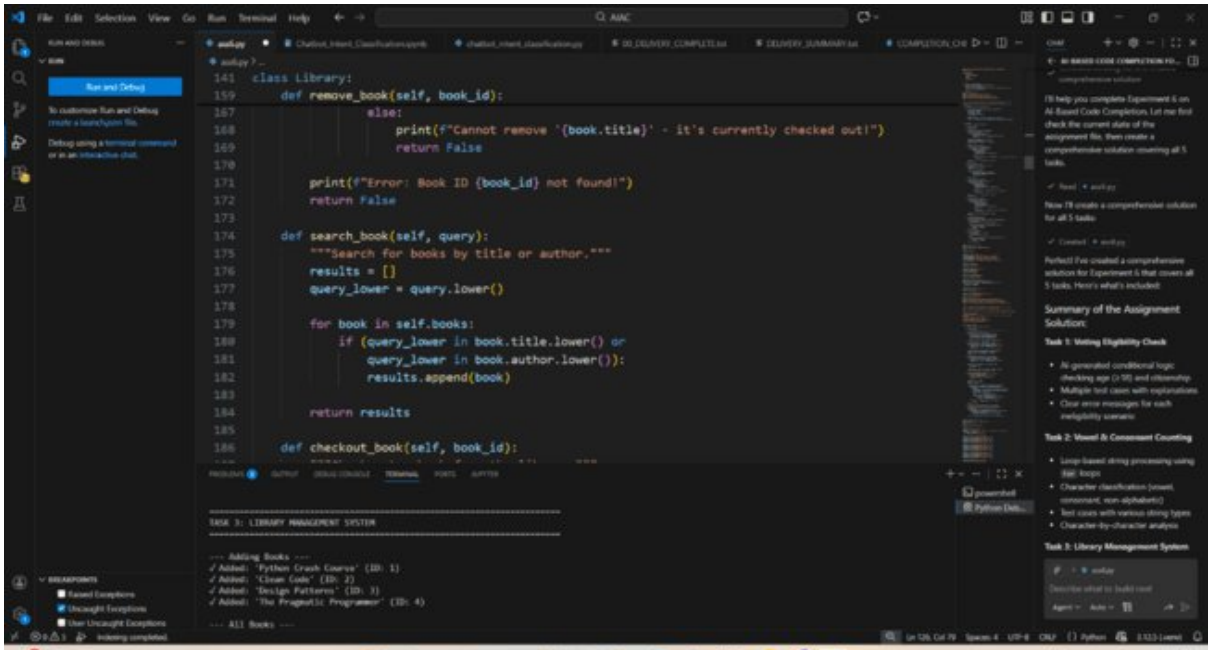
• AI-generated attendance logic.

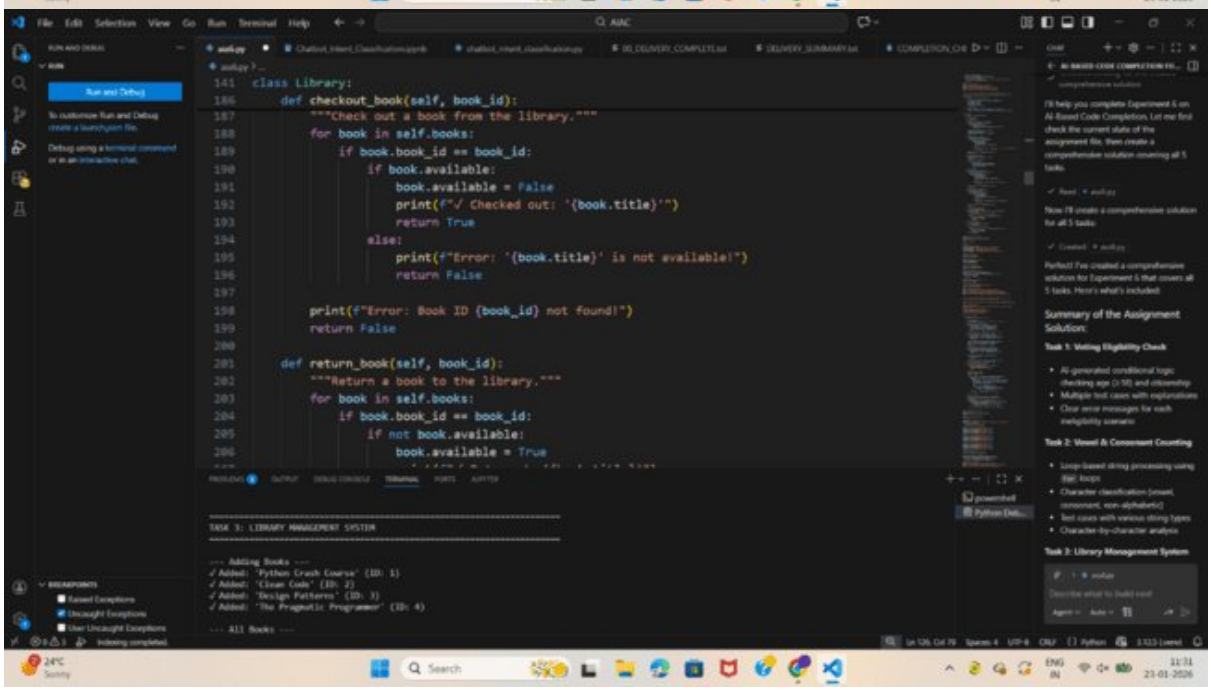
• Correct display of attendance.

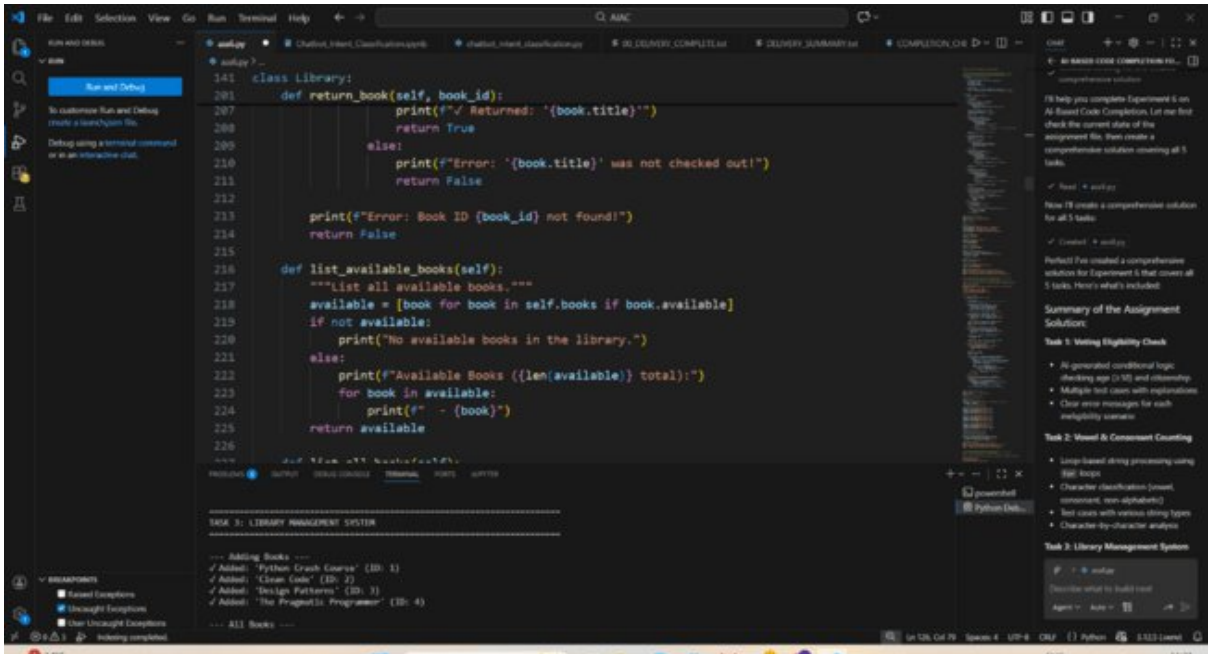
• Test cases.

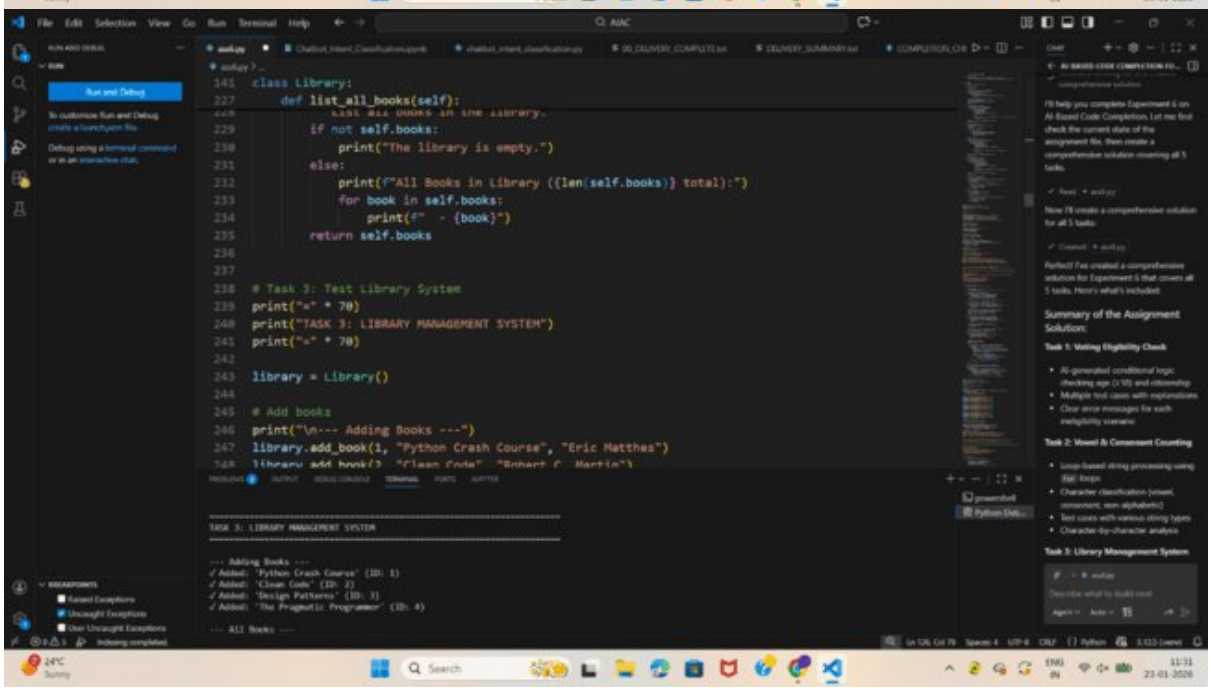

class Book:
"""Represents a book in the library."""
def __init__(self, book_id, title, author, available=True):
self.book_id = book_id
self.title = title
self.author = author
self.available = available
def __str__(self):
status = "Available" if self.available else "Checked Out"
return f"ID: {self.book_id} | '{self.title}' by {self.author} [{status}]"
class Library:
"""Manages a collection of books."""
def __init__(self):
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---

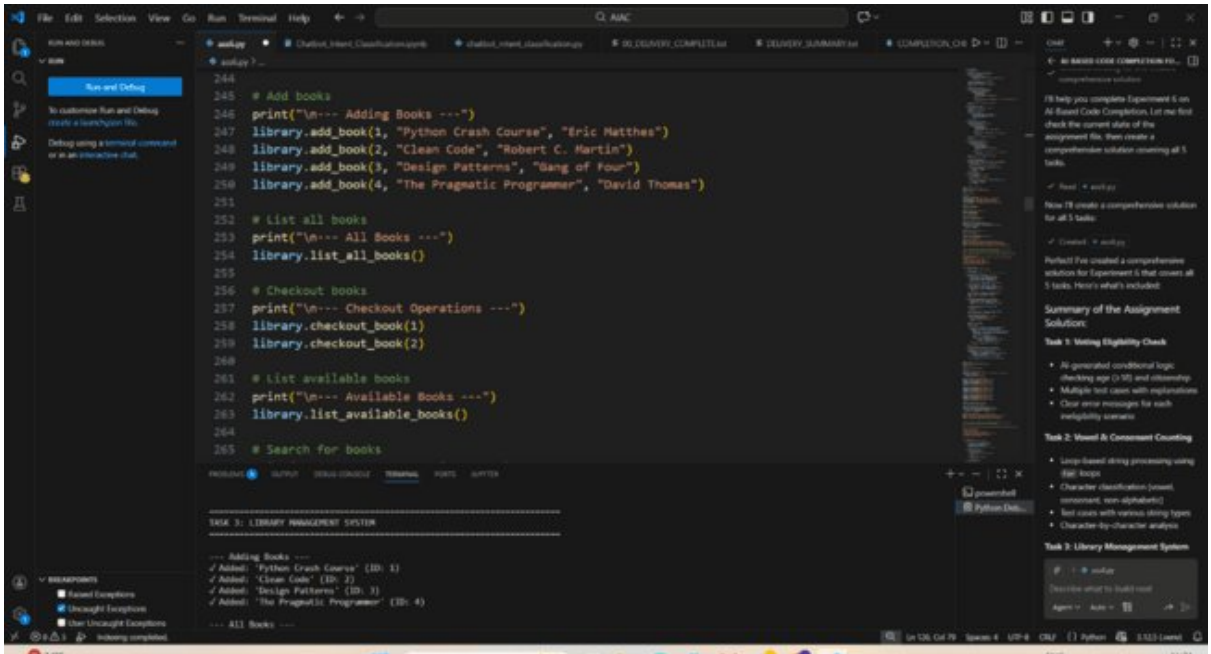

class Library:
def add_book(self, book_id, title, author):
"""Add a new book to the library."""
for book in self.books:
if book.book_id == book_id:
print(f"Error: Book ID {book_id} already exists!")
return False
new_book = Book(book_id, title, author)
self.books.append(new_book)
print(f"✓ Added: '{title}' (ID: {book_id})")
return True
def remove_book(self, book_id):
"""Remove a book from the library."""
for i, book in enumerate(self.books):
if book.book_id == book_id:
if book.available:
self.books.pop(i)
print(f"✓ Removed: '{book.title}'")
return True
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---


class Library:
def remove_book(self, book_id):
print(f"Cannot remove '{book.title}' - it's currently checked out!")
return False
print(f"Error: Book ID {book_id} not found!")
return False
def search_book(self, query):
"""Search for books by title or author."""
results = []
query_lower = query.lower()
for book in self.books:
if (query_lower in book.title.lower() or
query_lower in book.author.lower()):
results.append(book)
return results
def checkout_book(self, book_id):
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---


class Library:
def checkout_book(self, book_id):
"""Check out a book from the library."""
for book in self.books:
if book.book_id == book_id:
if book.available:
book.available = False
print(f"✓ Checked out: '{book.title}'")
return True
else:
print(f"Error: '{book.title}' is not available!")
return False
print(f"Error: Book ID {book_id} not found!")
return False
def return_book(self, book_id):
"""Return a book to the library."""
for book in self.books:
if book.book_id == book_id:
if not book.available:
book.available = True
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---

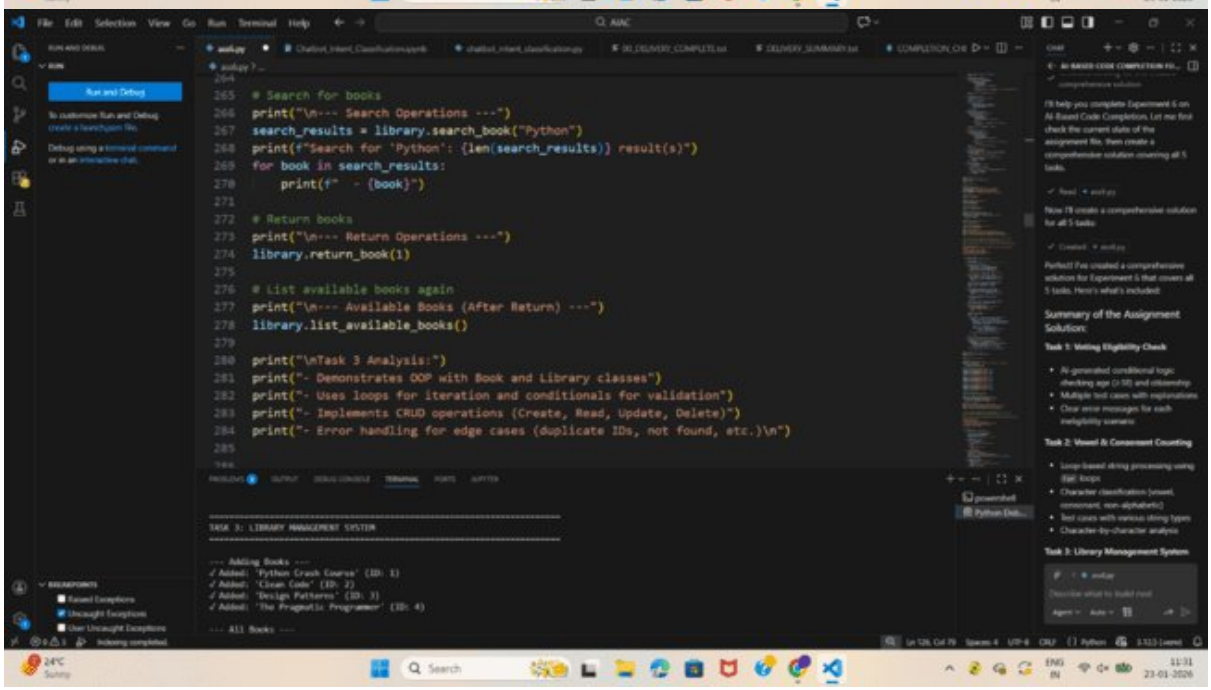

class Library:
def return_book(self, book_id):
print(f"✓ Returned: '{book.title}'")
return True
else:
print(f"Error: '{book.title}' was not checked out!")
return False
print(f"Error: Book ID {book_id} not found!")
return False
def list_available_books(self):
"""List all available books."""
available = [book for book in self.books if book.available]
if not available:
print("No available books in the library.")
else:
print(f"Available Books ({len(available)} total):")
for book in available:
print(f" - {book}")
return available
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books: ---


class Library:
def list_all_books(self):
if not self.books:
print("The library is empty.")
else:
print(f"All Books in Library ({len(self.books)} total):")
for book in self.books:
print(f" - {book}")
return self.books
# Task 3: Test Library System
print("=" * 70)
print("TASK 3: LIBRARY MANAGEMENT SYSTEM")
print("=" * 70)
library = Library()
# Add books
print("\n--- Adding Books ---")
library.add_book(1, "Python Crash Course", "Eric Matthes")
TASK 3: LIBRARY MANAGEMENT SYSTEM
--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books: ---



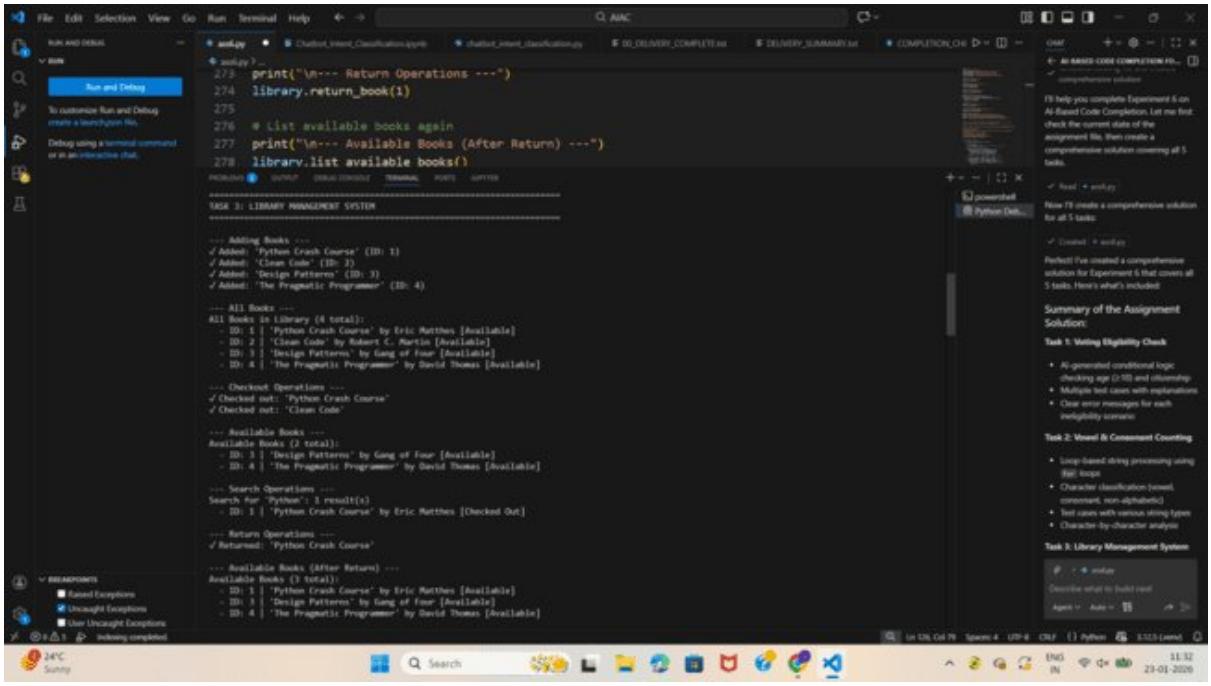
```
# Add books
print("\n--- Adding Books ---")
library.add_book(1, "Python Crash Course", "Eric Matthes")
library.add_book(2, "Clean Code", "Robert C. Martin")
library.add_book(3, "Design Patterns", "Gang of Four")
library.add_book(4, "The Pragmatic Programmer", "David Thomas")

# List all books
print("\n--- All Books ---")
library.list_all_books()

# Checkout books
print("\n--- Checkout Operations ---")
library.checkout_book(1)
library.checkout_book(2)

# List available books
print("\n--- Available Books ---")
library.list_available_books()

# Search for books
```

```
============================================================
TASK 3: LIBRARY MANAGEMENT SYSTEM
============================================================

--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)

--- All Books ---
```



```
# Search for books
print("\n--- Search Operations ---")
search_results = library.search_book("Python")
print(f"Search for 'Python': {len(search_results)} result(s)")
for book in search_results:
    print(f"  - {book}")

# Return books
print("\n--- Return Operations ---")
library.return_book(1)

# List available books again
print("\n--- Available Books (After Return) ---")
library.list_available_books()

print("\nTask 3 Analysis:")
print("- Demonstrates OOP with Book and Library classes")
print("- Uses loops for iteration and conditionals for validation")
print("- Implements CRUD operations (Create, Read, Update, Delete)")
print("- Error handling for edge cases (duplicate IDs, not found, etc.)\n")
```

```
============================================================
TASK 3: LIBRARY MANAGEMENT SYSTEM
============================================================

--- Adding Books ---
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)

--- All Books ---
```



Task Description #4 (AI-Assisted Code Completion for Class-

Based Attendance System)

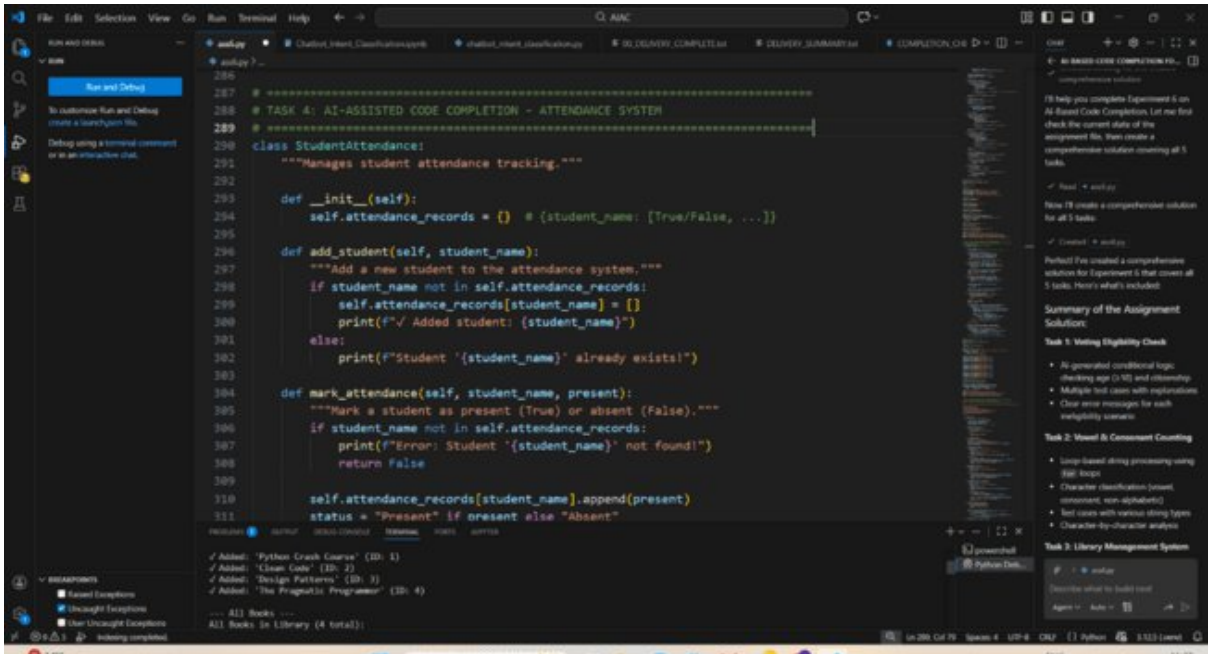
Task: Use an AI tool to generate an attendance management class.

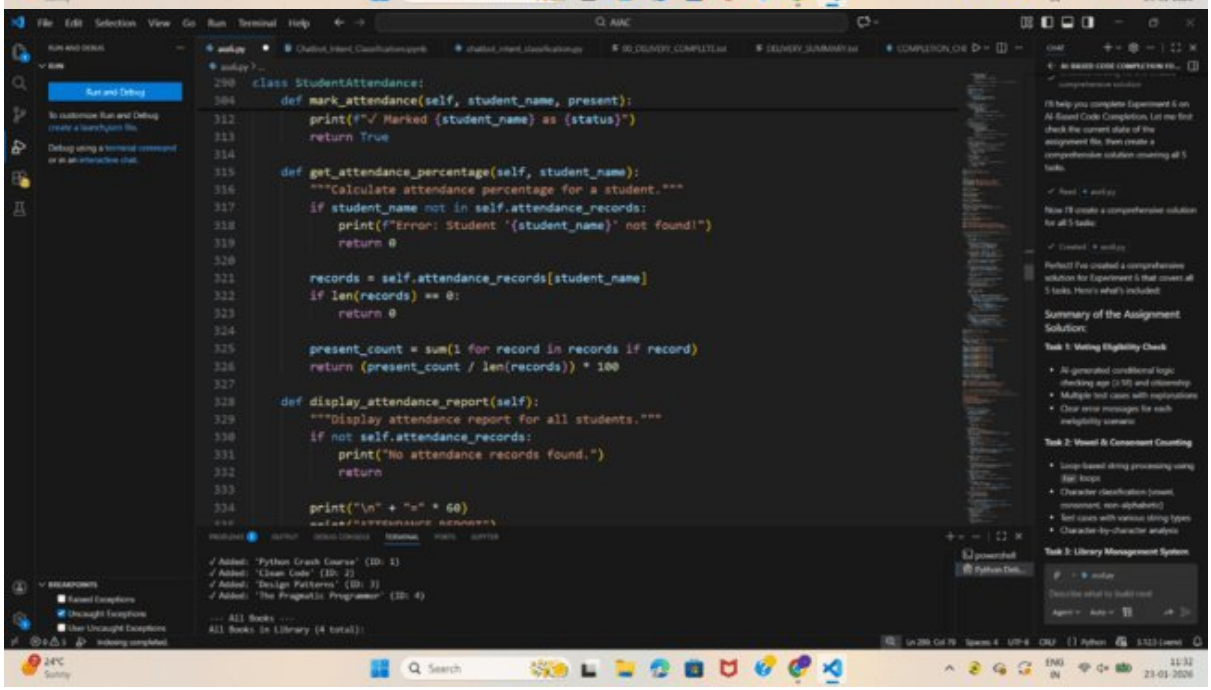
Prompt: “Generate a Python class to mark and display student

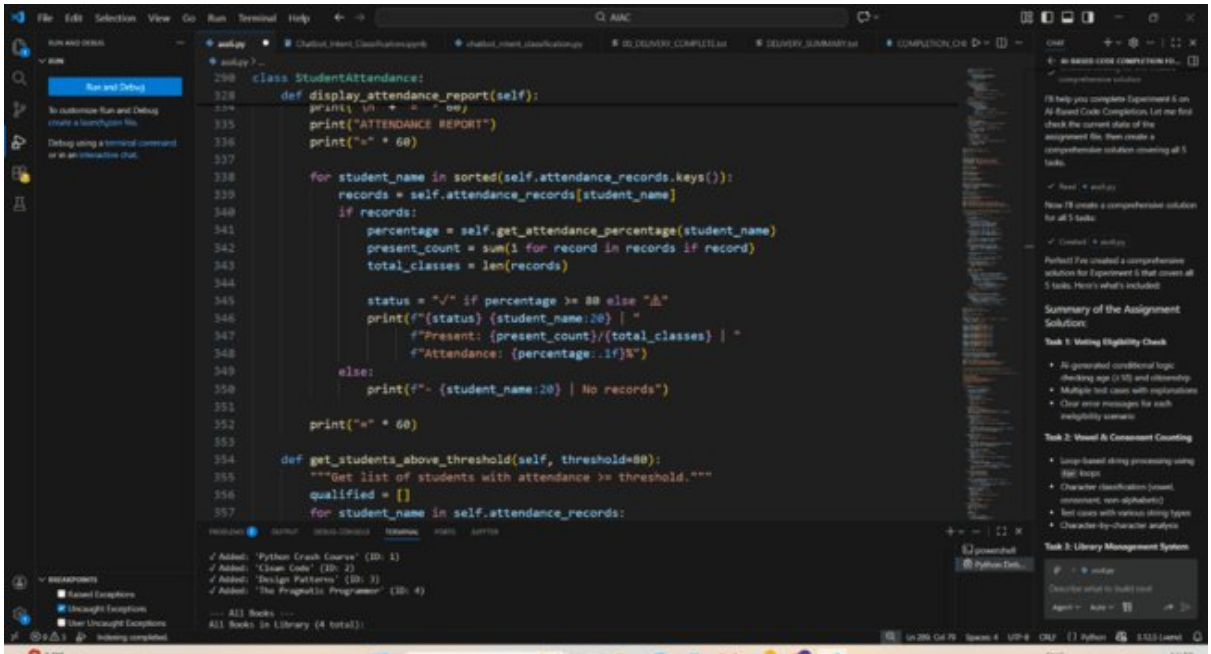
attendance using loops.”

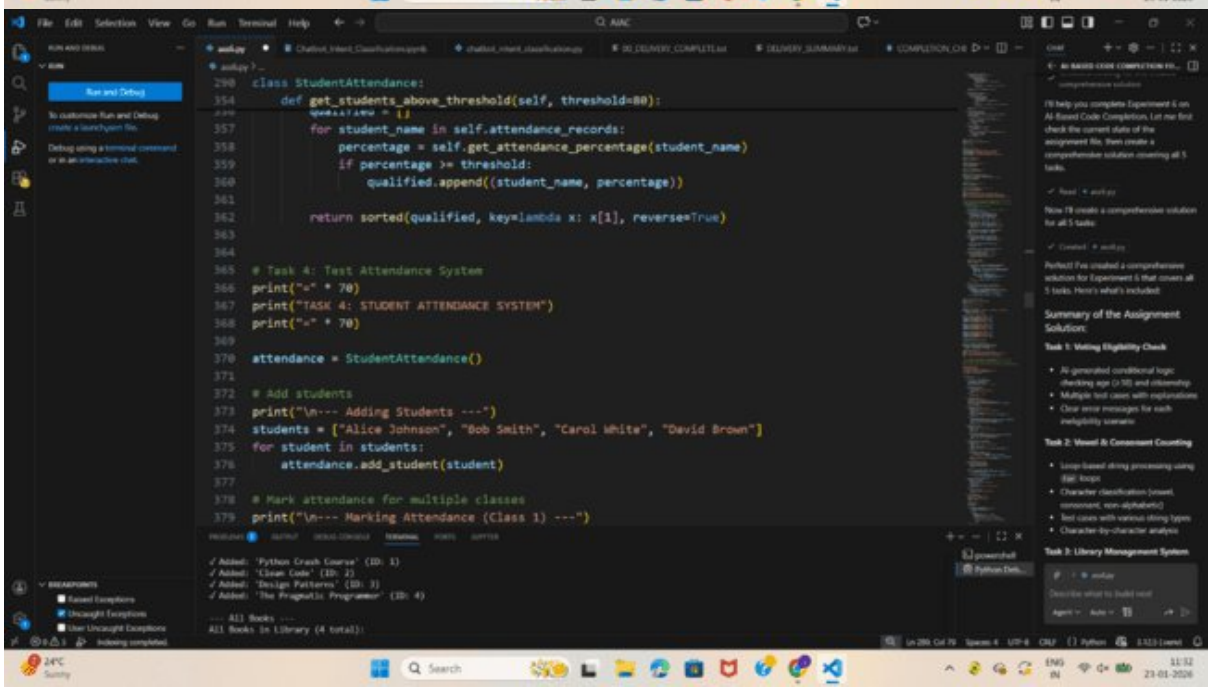
Expected Output:

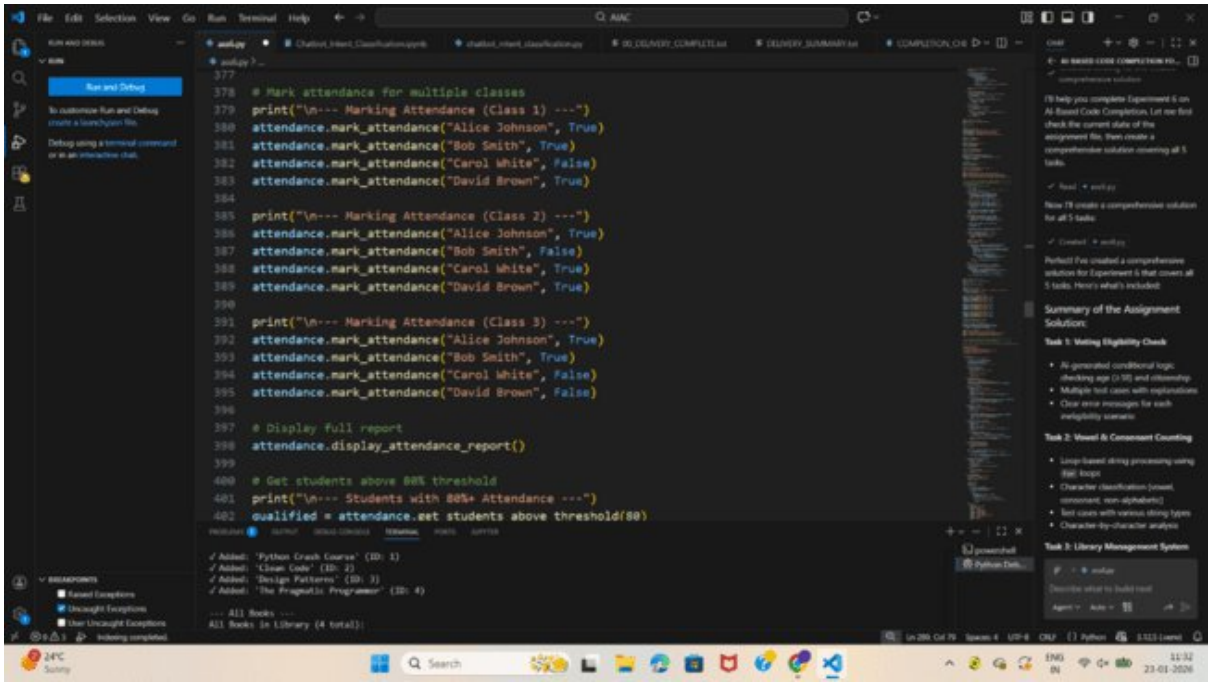
- AI-generated attendance logic.
- Correct display of attendance.
- Test cases.

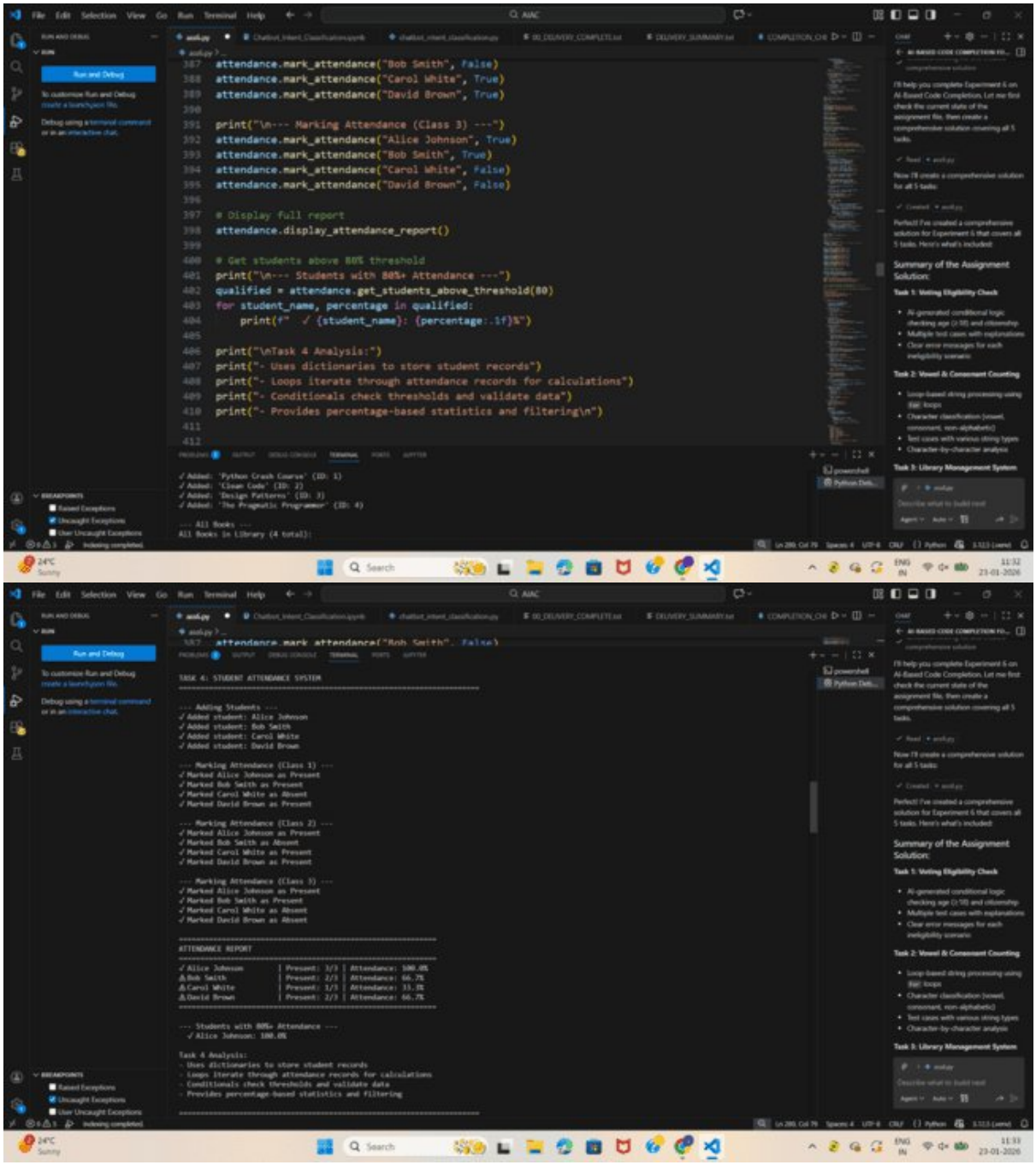

File Edit Selection View Go Run Terminal Help
# TASK 4: AI-ASSISTED CODE COMPLETION - ATTENDANCE SYSTEM
class StudentAttendance:
"""Manages student attendance tracking."""
def __init__(self):
self.attendance_records = {} # {student_name: [True/False, ...]}
def add_student(self, student_name):
"""Add a new student to the attendance system."""
if student_name not in self.attendance_records:
self.attendance_records[student_name] = []
print(f"✓ Added student: {student_name}")
else:
print(f"Student '{student_name}' already exists!")
def mark_attendance(self, student_name, present):
"""Mark a student as present (True) or absent (False)."""
if student_name not in self.attendance_records:
print(f"Error: Student '{student_name}' not found!")
return False
self.attendance_records[student_name].append(present)
status = "Present" if present else "Absent"
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---
All Books in Library (4 total):


class StudentAttendance:
def mark_attendance(self, student_name, present):
print(f"✓ Marked {student_name} as {status}")
return True
def get_attendance_percentage(self, student_name):
"""Calculate attendance percentage for a student."""
if student_name not in self.attendance_records:
print(f"Error: Student '{student_name}' not found!")
return 0
records = self.attendance_records[student_name]
if len(records) == 0:
return 0
present_count = sum(1 for record in records if record)
return (present_count / len(records)) * 100
def display_attendance_report(self):
"""Display attendance report for all students."""
if not self.attendance_records:
print("No attendance records found.")
return
print("\n" + "=" * 60)
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---
All Books in Library (4 total):


class StudentAttendance:
def display_attendance_report(self):
print("ATTENDANCE REPORT")
print("=" * 60)
for student_name in sorted(self.attendance_records.keys()):
records = self.attendance_records[student_name]
if records:
percentage = self.get_attendance_percentage(student_name)
present_count = sum(1 for record in records if record)
total_classes = len(records)
status = "✓" if percentage >= 80 else "⚠"
print(f"{status} {student_name:20} | "
f"Present: {present_count}/{total_classes} | "
f"Attendance: {percentage:.1f}%")
else:
print(f"- {student_name:20} | No records")
print("=" * 60)
def get_students_above_threshold(self, threshold=80):
"""Get list of students with attendance >= threshold."""
qualified = []
for student_name in self.attendance_records:
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---
All Books in Library (4 total):


class StudentAttendance:
def get_students_above_threshold(self, threshold=80):
for student_name in self.attendance_records:
percentage = self.get_attendance_percentage(student_name)
if percentage >= threshold:
qualified.append((student_name, percentage))
return sorted(qualified, key=lambda x: x[1], reverse=True)
# Task 4: Test Attendance System
print("=" * 70)
print("TASK 4: STUDENT ATTENDANCE SYSTEM")
print("=" * 70)
attendance = StudentAttendance()
# Add students
print("\n--- Adding Students ---")
students = ["Alice Johnson", "Bob Smith", "Carol White", "David Brown"]
for student in students:
attendance.add_student(student)
# Mark attendance for multiple classes
print("\n--- Marking Attendance (Class 1) ---")
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---
All Books in Library (4 total):


File Edit Selection View Go Run Terminal Help
AIAC
Run and Debug
# Mark attendance for multiple classes
print("\n--- Marking Attendance (Class 1) ---")
attendance.mark_attendance("Alice Johnson", True)
attendance.mark_attendance("Bob Smith", True)
attendance.mark_attendance("Carol White", False)
attendance.mark_attendance("David Brown", True)
print("\n--- Marking Attendance (Class 2) ---")
attendance.mark_attendance("Alice Johnson", True)
attendance.mark_attendance("Bob Smith", False)
attendance.mark_attendance("Carol White", True)
attendance.mark_attendance("David Brown", True)
print("\n--- Marking Attendance (Class 3) ---")
attendance.mark_attendance("Alice Johnson", True)
attendance.mark_attendance("Bob Smith", True)
attendance.mark_attendance("Carol White", False)
attendance.mark_attendance("David Brown", False)
# Display full report
attendance.display_attendance_report()
# Get students above 80% threshold
print("\n--- Students with 80%+ Attendance ---")
qualified = attendance.get_students_above_threshold(80)
✓ Added: 'Python Crash Course' (ID: 1)
✓ Added: 'Clean Code' (ID: 2)
✓ Added: 'Design Patterns' (ID: 3)
✓ Added: 'The Pragmatic Programmer' (ID: 4)
--- All Books ---
All Books in Library (4 total):
Summary of the Assignment Solution:
Task 1: Voting Eligibility Check
Task 2: Vowel & Consonant Counting
Task 3: Library Management System
24°C
Sunny
Search
ENG
IN
11:32
23-01-2026



Task Description #5 (AI-Based Code Completion for Conditional

Menu Navigation)

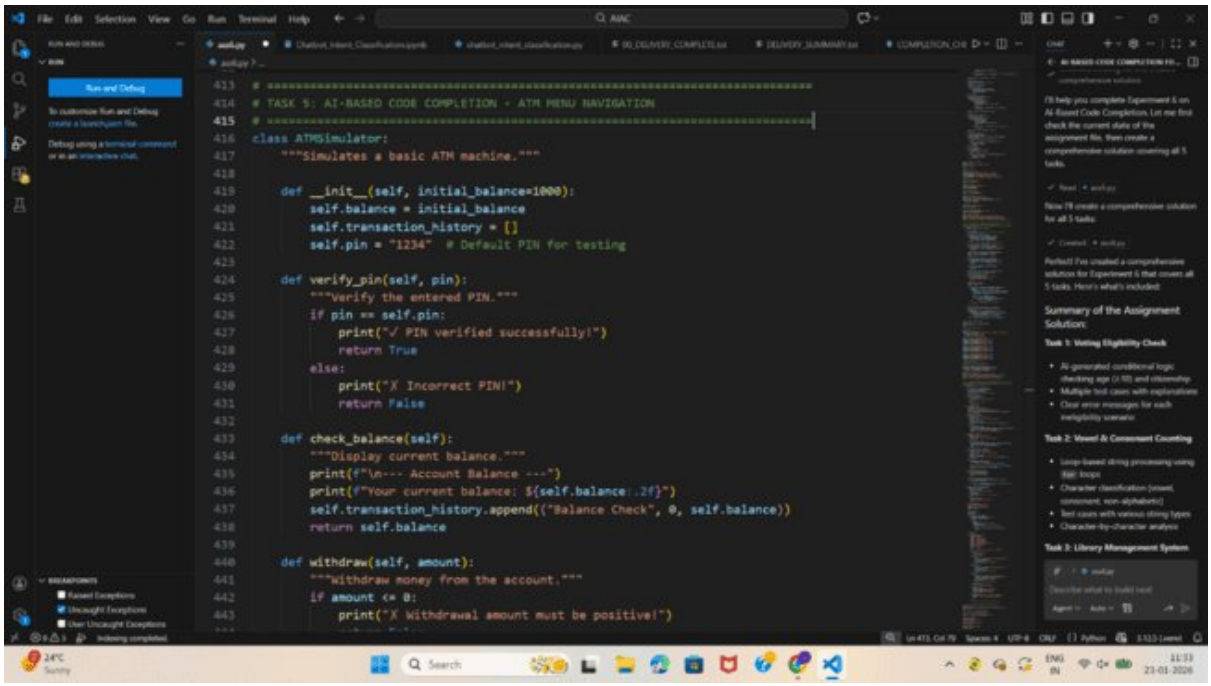
Task: Use an AI tool to complete a navigation menu.

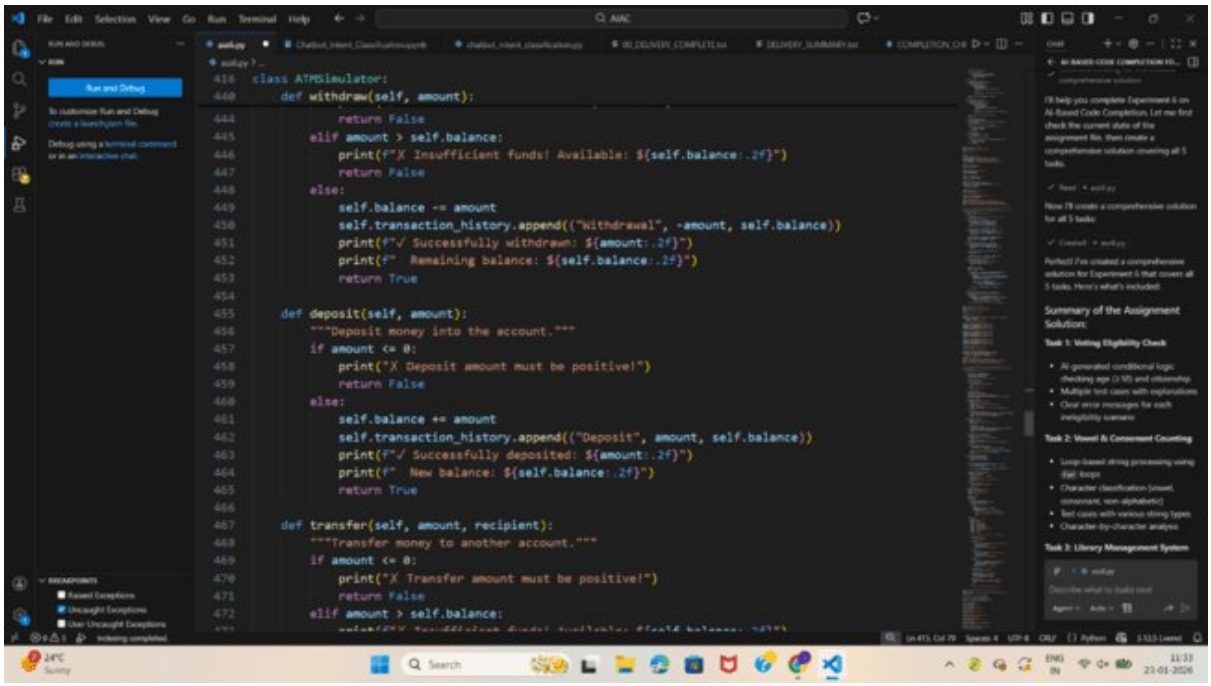
Prompt: “Generate a Python program using loops and conditionals

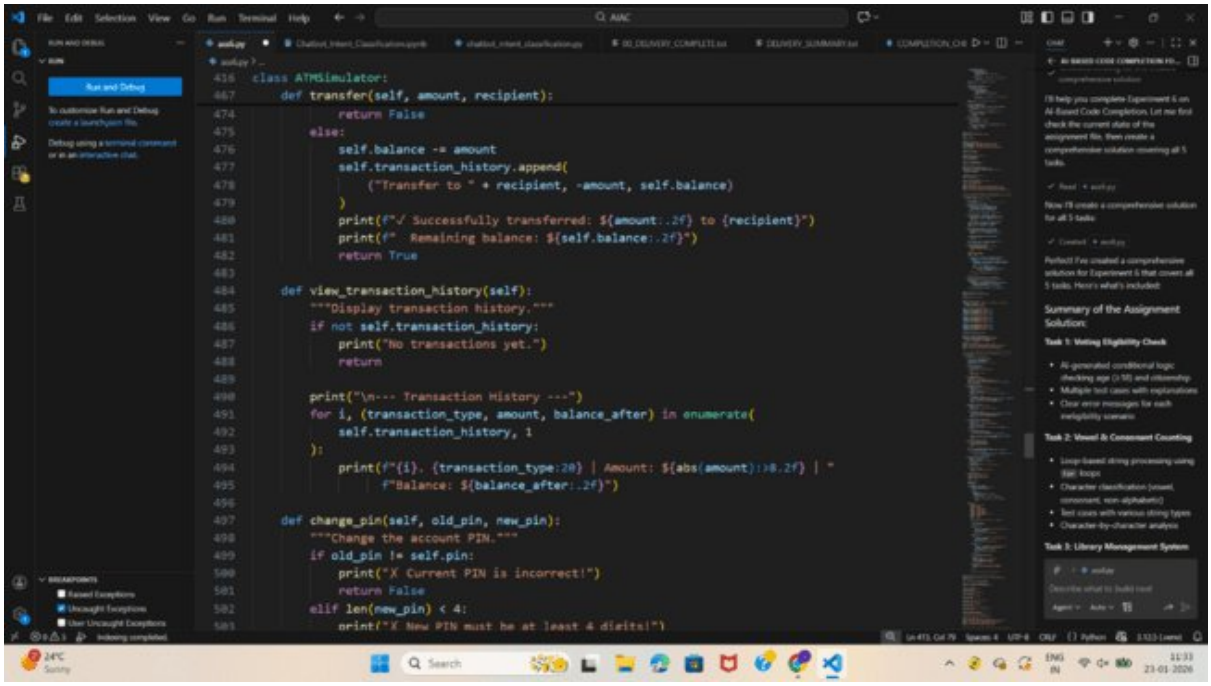
to simulate an ATM menu.”

Expected Output:

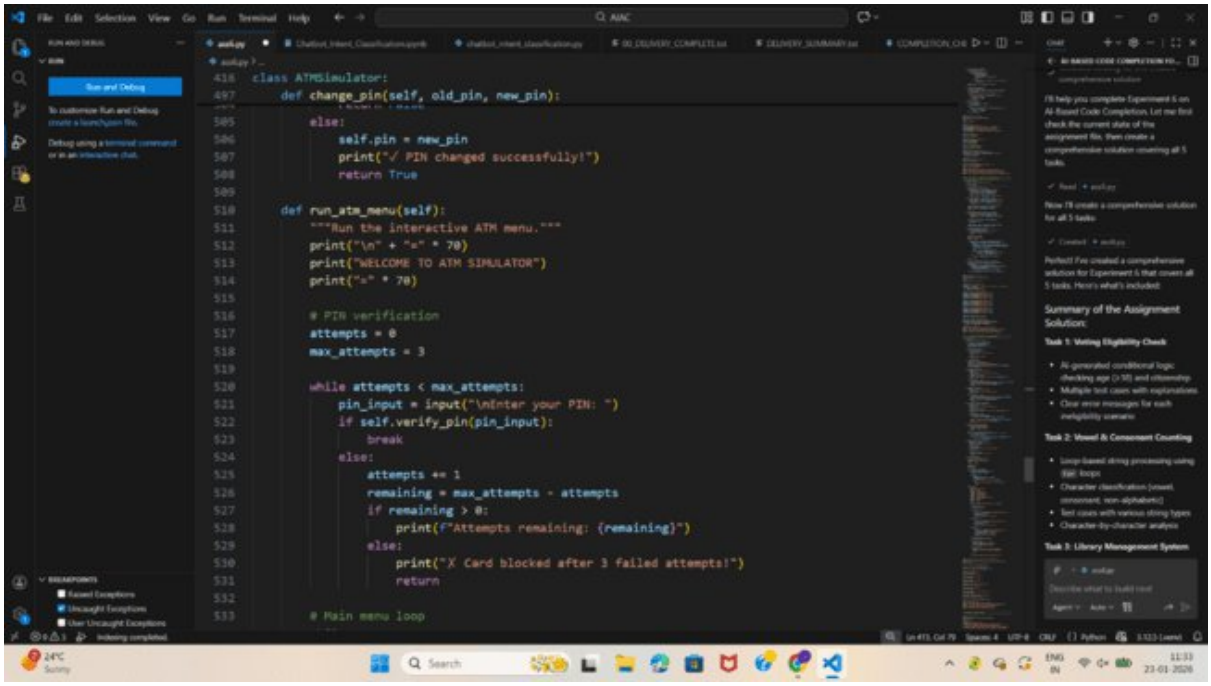
- AI-generated menu logic.
- Correct option handling.
- Output verification.



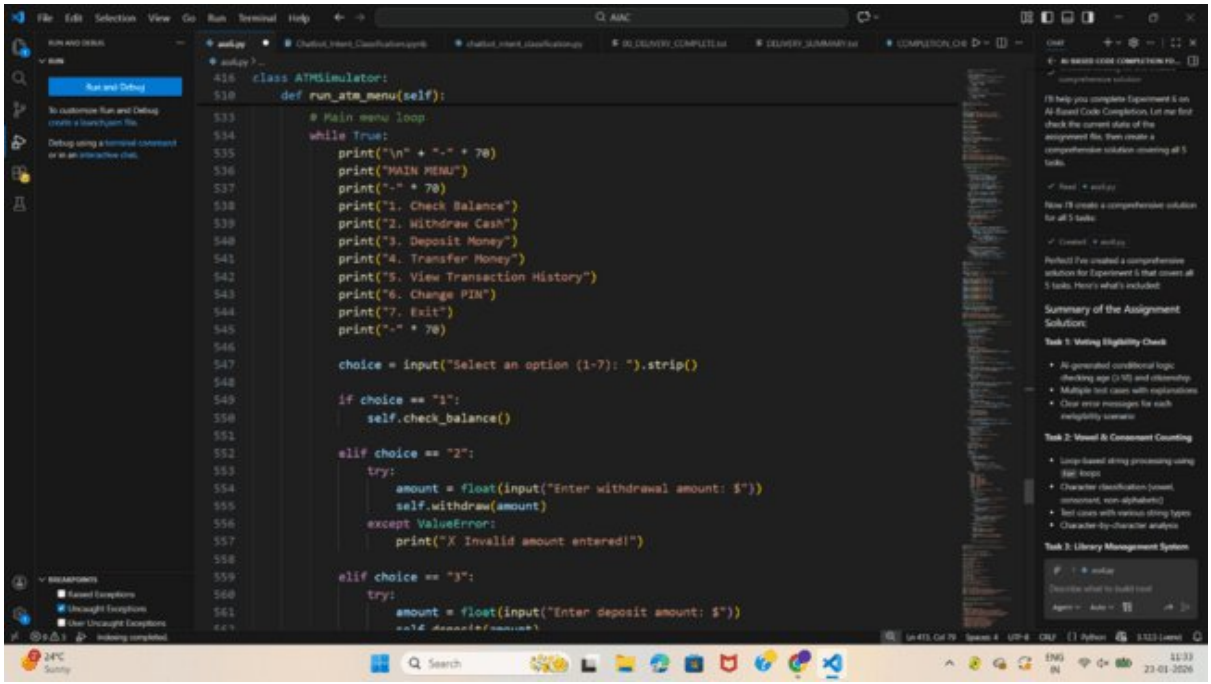




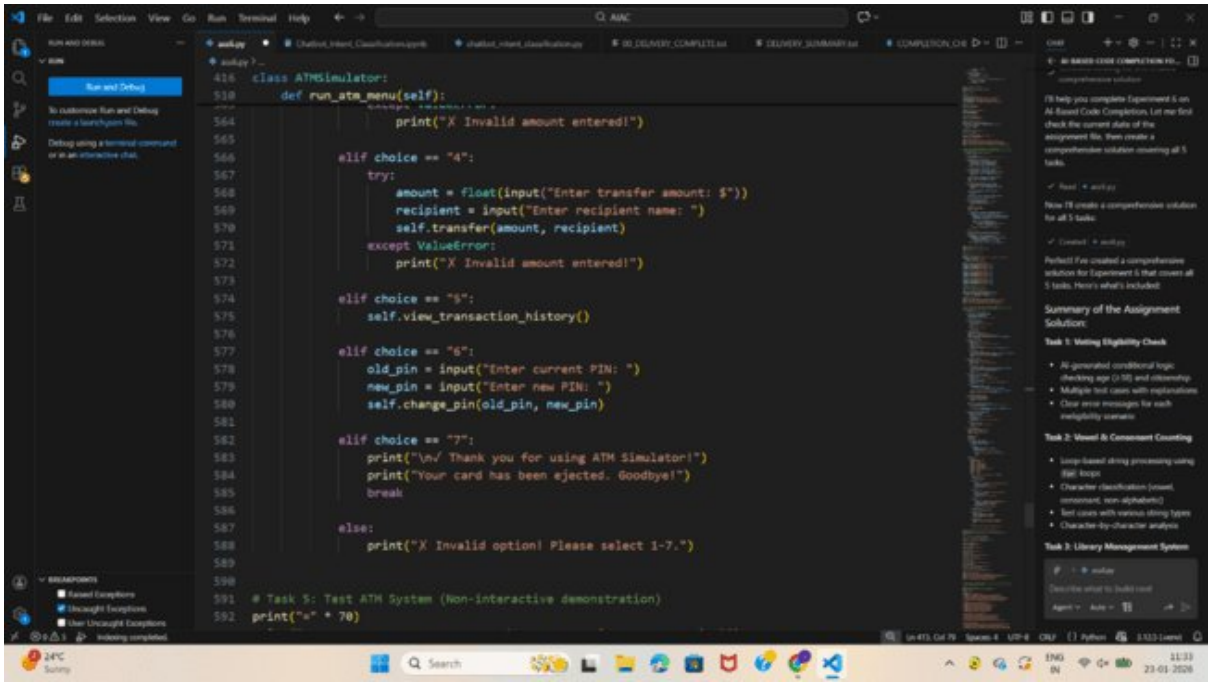
```
class ATMSimulator:
    def transfer(self, amount, recipient):
                return False
            else:
                self.balance -= amount
                self.transaction_history.append(
                    ("Transfer to " + recipient, -amount, self.balance)
                )
                print(f"✓ Successfully transferred: ${amount:.2f} to {recipient}")
                print(f"  Remaining balance: ${self.balance:.2f}")
                return True

    def view_transaction_history(self):
        """Display transaction history."""
        if not self.transaction_history:
            print("No transactions yet.")
            return

        print("\n--- Transaction History ---")
        for i, (transaction_type, amount, balance_after) in enumerate(
            self.transaction_history, 1
        ):
            print(f"{i}. {transaction_type:20} | Amount: ${abs(amount):8.2f} | "
                  f"Balance: ${balance_after:.2f}")

    def change_pin(self, old_pin, new_pin):
        """Change the account PIN."""
        if old_pin != self.pin:
            print("✗ Current PIN is incorrect!")
            return False
        elif len(new_pin) < 4:
            print("✗ New PIN must be at least 4 digits!")
```



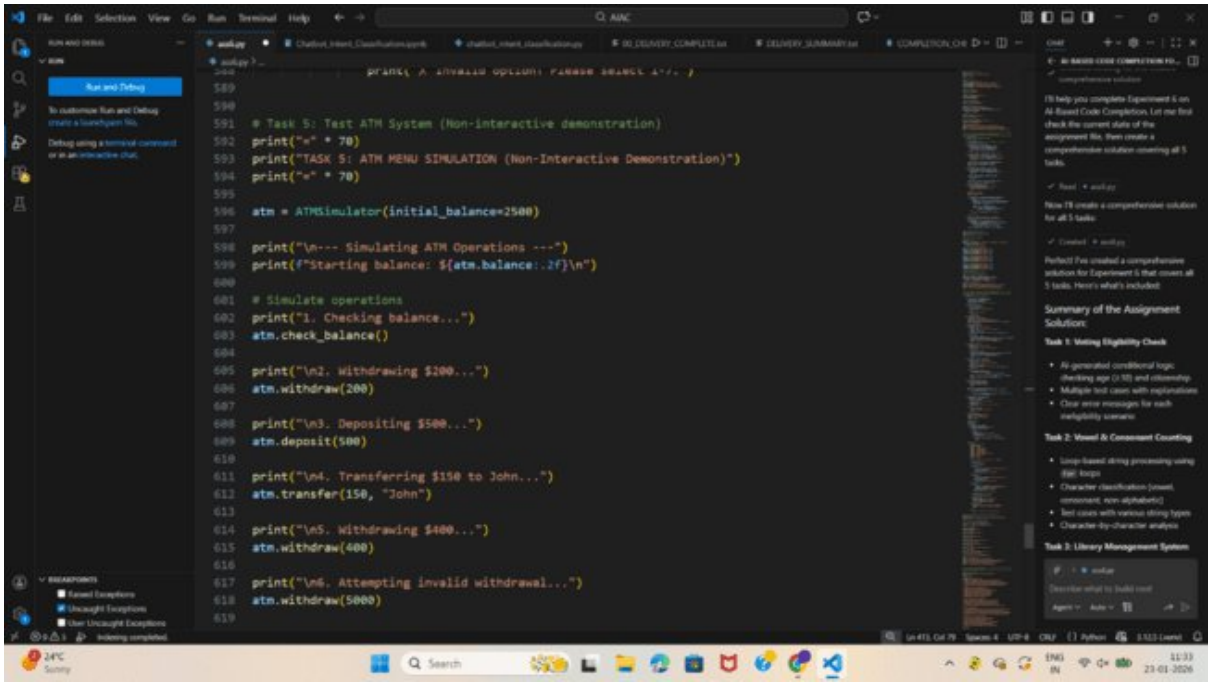
```
class ATMSimulator:
    def change_pin(self, old_pin, new_pin):
            else:
                self.pin = new_pin
                print("✓ PIN changed successfully!")
                return True

    def run_atm_menu(self):
        """Run the interactive ATM menu."""
        print("\n" + "=" * 70)
        print("WELCOME TO ATM SIMULATOR")
        print("=" * 70)

        # PIN verification
        attempts = 0
        max_attempts = 3

        while attempts < max_attempts:
            pin_input = input("\nEnter your PIN: ")
            if self.verify_pin(pin_input):
                break
            else:
                attempts += 1
                remaining = max_attempts - attempts
                if remaining > 0:
                    print(f"Attempts remaining: {remaining}")
                else:
                    print("✗ Card blocked after 3 failed attempts!")
                    return

        # Main menu loop
```

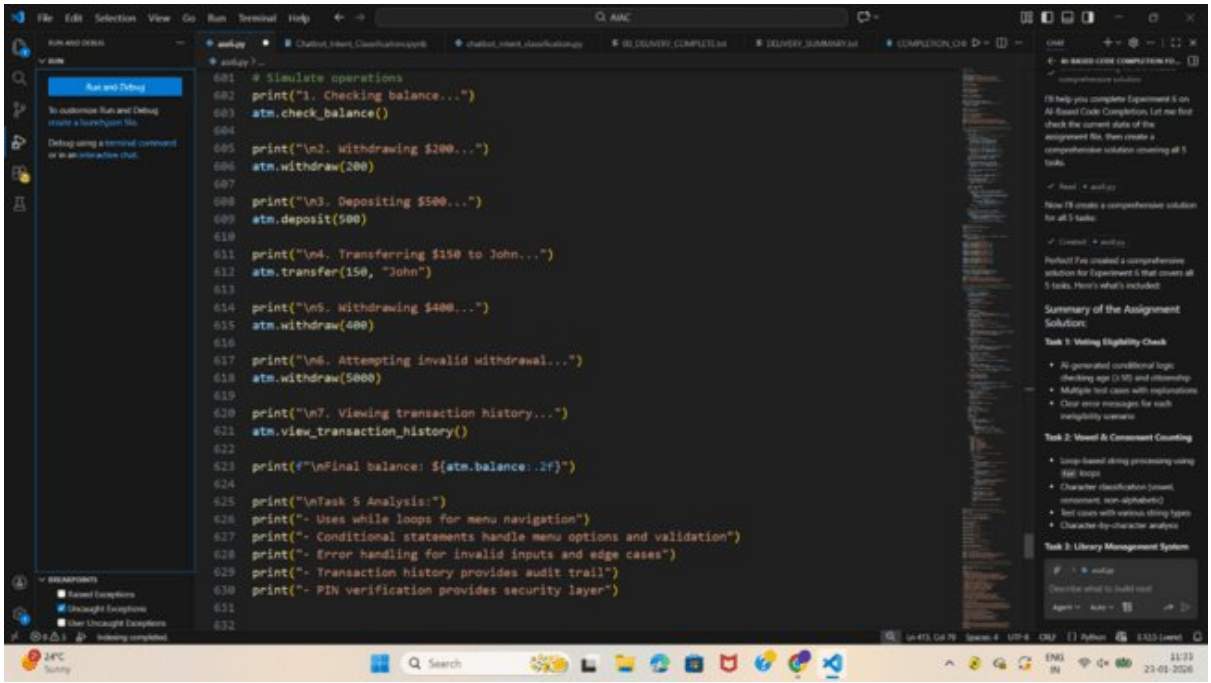


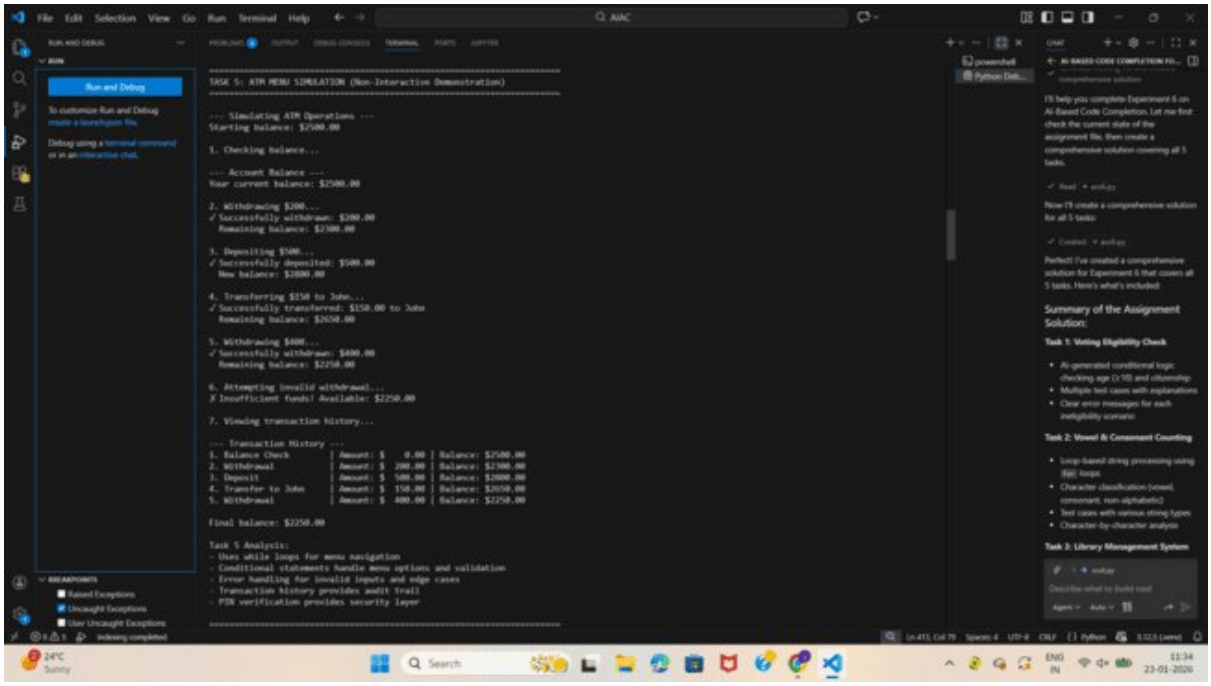
```
class ATMSimulator:
    def run_atm_menu(self):
        # Main menu loop
        while True:
            print("\n" + "-" * 70)
            print("MAIN MENU")
            print("-" * 70)
            print("1. Check Balance")
            print("2. Withdraw Cash")
            print("3. Deposit Money")
            print("4. Transfer Money")
            print("5. View Transaction History")
            print("6. Change PIN")
            print("7. Exit")
            print("-" * 70)

            choice = input("Select an option (1-7): ").strip()

            if choice == "1":
                self.check_balance()

            elif choice == "2":
                try:
                    amount = float(input("Enter withdrawal amount: $"))
                    self.withdraw(amount)
                except ValueError:
                    print("X Invalid amount entered!")

            elif choice == "3":
                try:
                    amount = float(input("Enter deposit amount: $"))
```



```
class ATMSimulator:
    def run_atm_menu(self):
                    print("X Invalid amount entered!")

            elif choice == "4":
                try:
                    amount = float(input("Enter transfer amount: $"))
                    recipient = input("Enter recipient name: ")
                    self.transfer(amount, recipient)
                except ValueError:
                    print("X Invalid amount entered!")

            elif choice == "5":
                self.view_transaction_history()

            elif choice == "6":
                old_pin = input("Enter current PIN: ")
                new_pin = input("Enter new PIN: ")
                self.change_pin(old_pin, new_pin)

            elif choice == "7":
                print("\n√ Thank you for using ATM Simulator!")
                print("Your card has been ejected. Goodbye!")
                break

            else:
                print("X Invalid option! Please select 1-7.")


# Task 5: Test ATM System (Non-interactive demonstration)
print("=" * 70)
```


print("❌ Invalid option! Please select 1-7.")
# Task 5: Test ATM System (Non-interactive demonstration)
print("=" * 70)
print("TASK 5: ATM MENU SIMULATION (Non-Interactive Demonstration)")
print("=" * 70)
atm = ATMSimulator(initial_balance=2500)
print("\n--- Simulating ATM Operations ---")
print(f"Starting balance: ${atm.balance:.2f}\n")
# Simulate operations
print("1. Checking balance...")
atm.check_balance()
print("\n2. Withdrawing $200...")
atm.withdraw(200)
print("\n3. Depositing $500...")
atm.deposit(500)
print("\n4. Transferring $150 to John...")
atm.transfer(150, "John")
print("\n5. Withdrawing $400...")
atm.withdraw(400)
print("\n6. Attempting invalid withdrawal...")
atm.withdraw(5000)


# Simulate operations
print("1. Checking balance...")
atm.check_balance()
print("\n2. Withdrawing $200...")
atm.withdraw(200)
print("\n3. Depositing $500...")
atm.deposit(500)
print("\n4. Transferring $150 to John...")
atm.transfer(150, "John")
print("\n5. Withdrawing $400...")
atm.withdraw(400)
print("\n6. Attempting invalid withdrawal...")
atm.withdraw(5000)
print("\n7. Viewing transaction history...")
atm.view_transaction_history()
print(f"\nFinal balance: ${atm.balance:.2f}")
print("\nTask 5 Analysis:")
print("- Uses while loops for menu navigation")
print("- Conditional statements handle menu options and validation")
print("- Error handling for invalid inputs and edge cases")
print("- Transaction history provides audit trail")
print("- PIN verification provides security layer")



Summary of the Assignment Solution:

Task 1: Voting Eligibility Check

- AI-generated conditional logic checking age (≥18) and citizenship
- Multiple test cases with explanations
- Clear error messages for each ineligibility scenario

Task 2: Vowel & Consonant Counting

- Loop-based string processing using for loops
- Character classification (vowel, consonant, non-alphabetic)
- Test cases with various string types
- Character-by-character analysis

Task 3: Library Management System

- Complete OOP implementation
  with Book and Library classes
- Features: add, remove, search, checkout, return books
- Demonstrates CRUD operations and error handling
- Real-world example of classes + loops + conditionals

Task 4: Student Attendance System

- StudentAttendance class for tracking attendance
- Methods to add students, mark attendance, calculate percentages
- Attendance report with statistics
- Filtering students by attendance threshold (80%+)

Task 5: ATM Menu Simulation

- ATMSimulator class with full banking operations
- Operations: balance check, withdraw, deposit, transfer
- PIN verification and transaction history
- Non-interactive demonstration with realistic scenarios

Plus:

- Comprehensive documentation and explanations for each task
- Test cases demonstrating correctness
- Analysis of each solution's key features

- **Reflection on AI-Assisted Coding** covering strengths, improvements, ethical considerations, and best practices

The file is ready to run and demonstrates all the learning objectives (LO1-LO5) with proper code interpretation, error identification, optimization, and ethical considerations.