AI ASSISTANT CODING
ASSIGNMENT-6.5

Name: D. Divya Sri

HT. No: 2303A51323

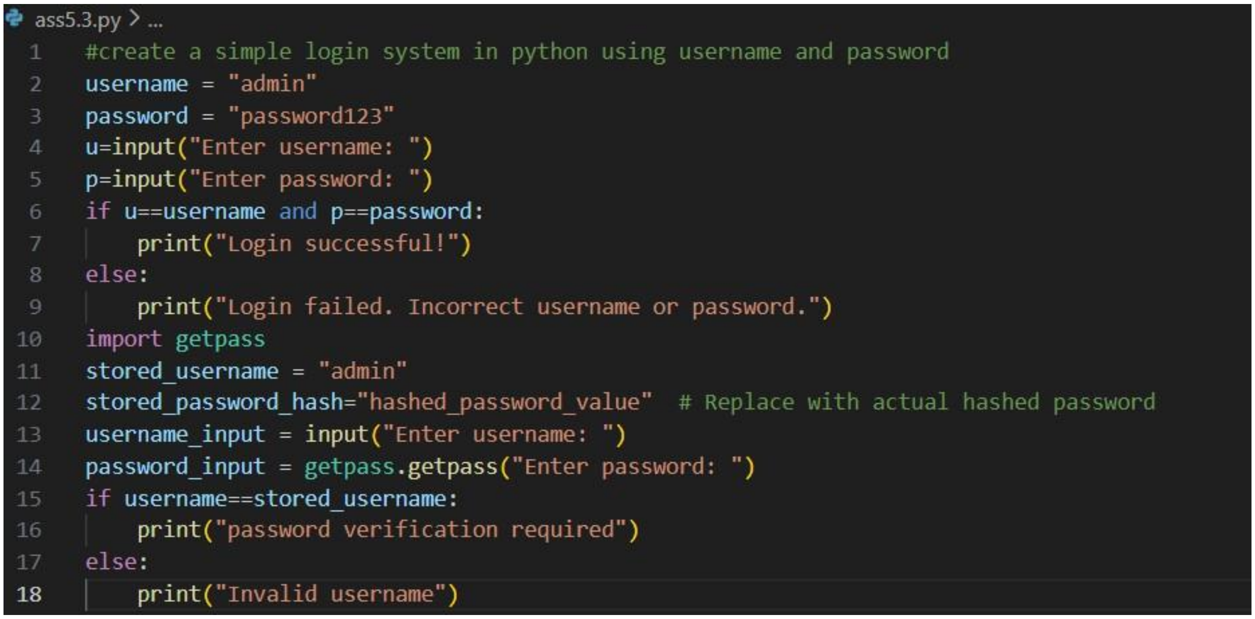
Batch: 19

Task 1: AI-Based Code Completion for Conditional Eligibility Check

Use an AI tool to generate eligibility logic.

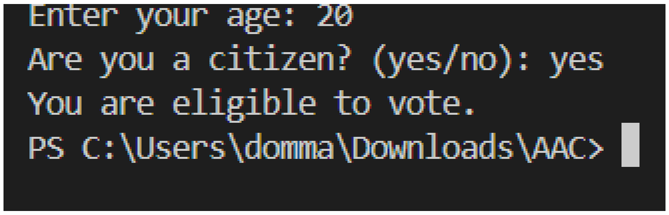
Prompt: # Generate Python code to check voting eligibility based on age and citizenship.

Code:



```
#create a simple login system in python using username and password
username = "admin"
password = "password123"
u=input("Enter username: ")
p=input("Enter password: ")
if u==username and p==password:
    print("Login successful!")
else:
    print("Login failed. Incorrect username or password.")
import getpass
stored_username = "admin"
stored_password_hash="hashed_password_value"  # Replace with actual hashed password
username_input = input("Enter username: ")
password_input = getpass.getpass("Enter password: ")
if username==stored_username:
    print("password verification required")
else:
    print("Invalid username")
```

Result:



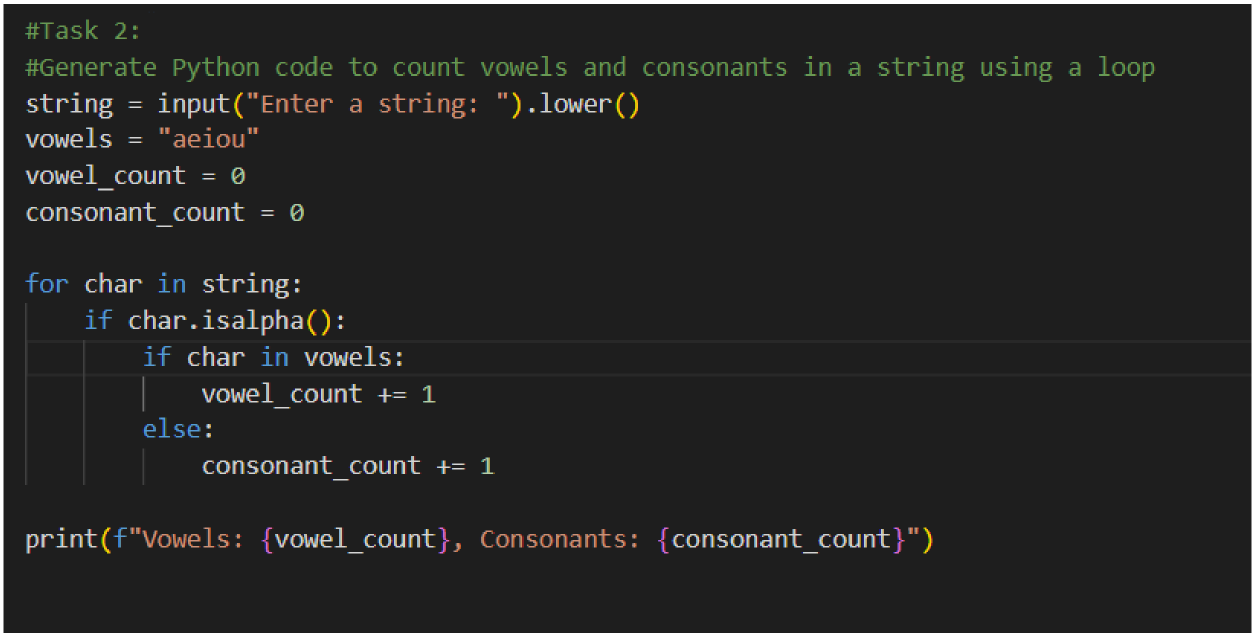
```
Enter your age: 20
Are you a citizen? (yes/no): yes
You are eligible to vote.
PS C:\Users\domma\Downloads\AAC>
```

Observation:

The AI-generated voting eligibility program determines eligibility based solely on age and citizenship. While the logic correctly enforces the minimum age requirement and citizenship condition, it lacks input validation and does not handle invalid or unexpected user inputs. Additionally, the system does not account for exceptional cases or regional voting rules, limiting its reliability and suitability for real-world.

Task 2: AI-Based Code Completion for Loop-Based String Processing

Use an AI tool to process strings using loops.
Prompt: # Generate Python code to count vowels and consonants in a string using a loop
Code:

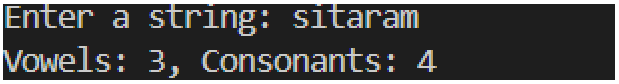


```python
#Task 2:
#Generate Python code to count vowels and consonants in a string using a loop
string = input("Enter a string: ").lower()
vowels = "aeiou"
vowel_count = 0
consonant_count = 0

for char in string:
    if char.isalpha():
        if char in vowels:
            vowel_count += 1
        else:
            consonant_count += 1

print(f"Vowels: {vowel_count}, Consonants: {consonant_count}")
```

Result:



```
Enter a string: sitaram
Vowels: 3, Consonants: 4
```

Observation:

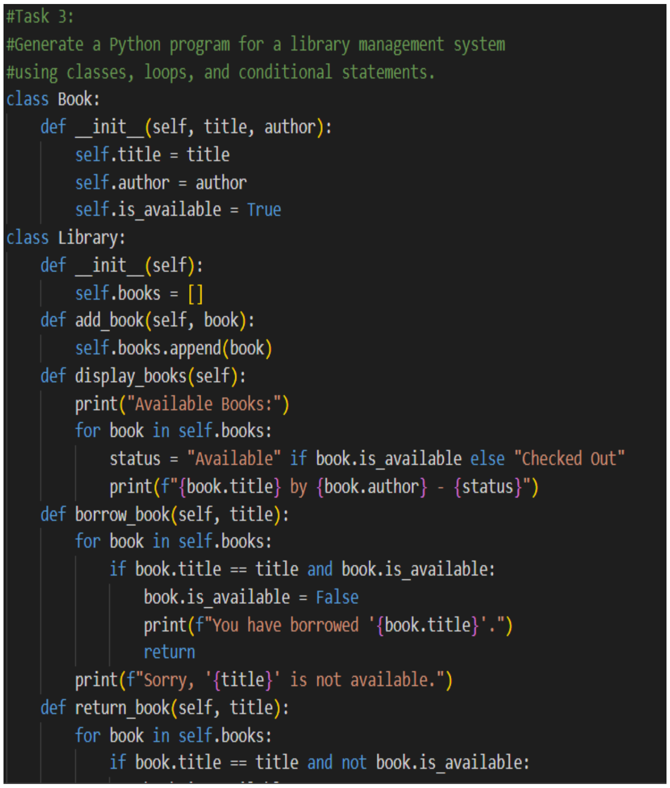
The AI-generated program correctly counts the number of vowels and consonants in a given string using a loop. It converts the input to lowercase to ensure case-insensitive comparison and uses isalpha() to ignore non-alphabetic characters. However, the program does not handle special cases such as empty input or accented characters, which may limit its accuracy in broader real-world scenario.

Task3: AI-Assisted Code Completion Reflection Task

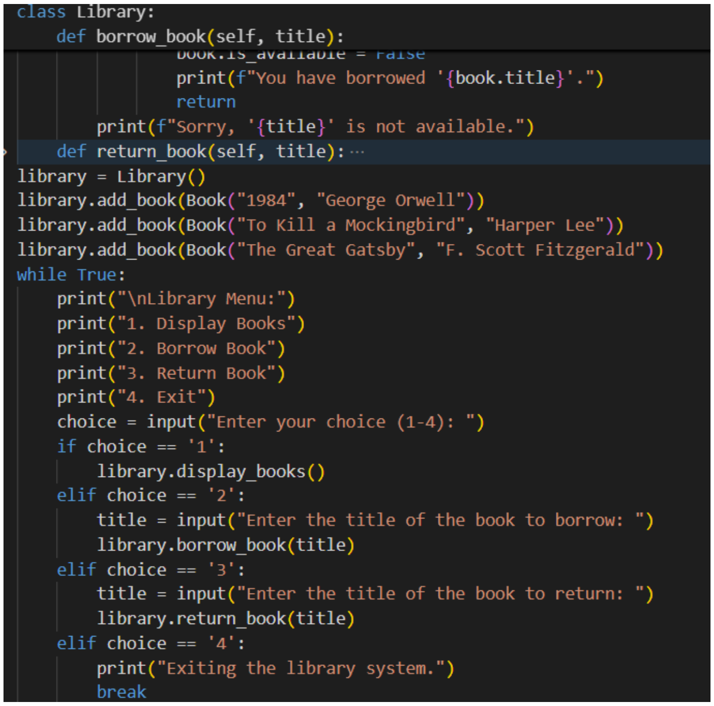
The purpose of this task is to Use an AI tool to generate a complete program using classes, loops, and conditionals.

Prompt: # Generate a Python program for a library management system

Code:

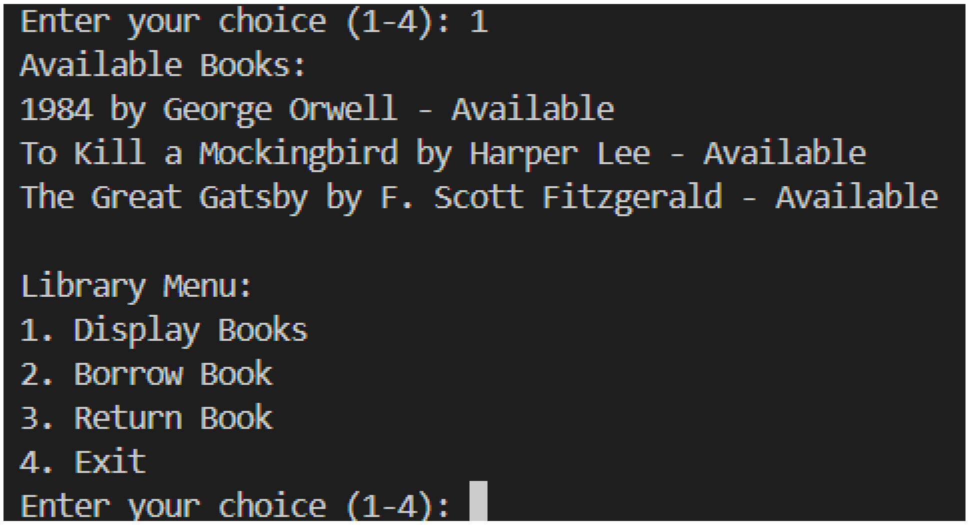


```python
#Task 3:
#Generate a Python program for a library management system
#using classes, loops, and conditional statements.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True
class Library:
    def __init__(self):
        self.books = []
    def add_book(self, book):
        self.books.append(book)
    def display_books(self):
        print("Available Books:")
        for book in self.books:
            status = "Available" if book.is_available else "Checked Out"
            print(f"{book.title} by {book.author} - {status}")
    def borrow_book(self, title):
        for book in self.books:
            if book.title == title and book.is_available:
                book.is_available = False
                print(f"You have borrowed '{book.title}'.")
                return
        print(f"Sorry, '{title}' is not available.")
    def return_book(self, title):
        for book in self.books:
            if book.title == title and not book.is_available:
```



```python
class Library:
    def borrow_book(self, title):
                book.is_available = False
                print(f"You have borrowed '{book.title}'.")
                return
        print(f"Sorry, '{title}' is not available.")
    def return_book(self, title):···
library = Library()
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.add_book(Book("The Great Gatsby", "F. Scott Fitzgerald"))
while True:
    print("\nLibrary Menu:")
    print("1. Display Books")
    print("2. Borrow Book")
    print("3. Return Book")
    print("4. Exit")
    choice = input("Enter your choice (1-4): ")
    if choice == '1':
        library.display_books()
    elif choice == '2':
        title = input("Enter the title of the book to borrow: ")
        library.borrow_book(title)
    elif choice == '3':
        title = input("Enter the title of the book to return: ")
        library.return_book(title)
    elif choice == '4':
        print("Exiting the library system.")
        break
```

Result:



```
Enter your choice (1-4): 1
Available Books:
1984 by George Orwell - Available
To Kill a Mockingbird by Harper Lee - Available
The Great Gatsby by F. Scott Fitzgerald - Available

Library Menu:
1. Display Books
2. Borrow Book
3. Return Book
4. Exit
Enter your choice (1-4): 
```

Observation:

The AI-generated library management system effectively demonstrates object-oriented programming by using separate Book and Library classes to manage books and their availability. It provides basic functionalities such as adding books, displaying available books, borrowing, and returning books through a menu-driven interface. However, the system lacks input validation, does not handle duplicate book titles, and does not persist data between runs, which limits its scalability and suitability for real-world library applications.

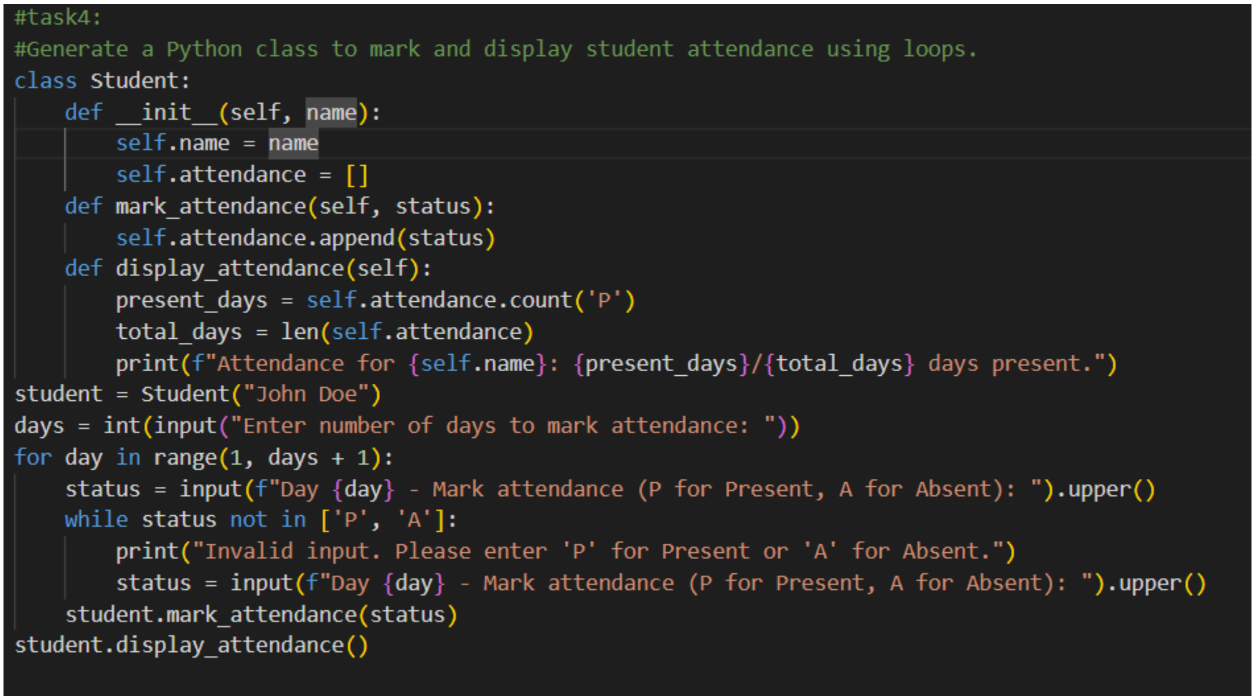
Task 4: AI-Assisted Code Completion for Class-Based Attendance System

The purpose of this task is use an AI tool to generate an attendance management class.

Prompt: # Generate a Python class to mark and display student attendance using

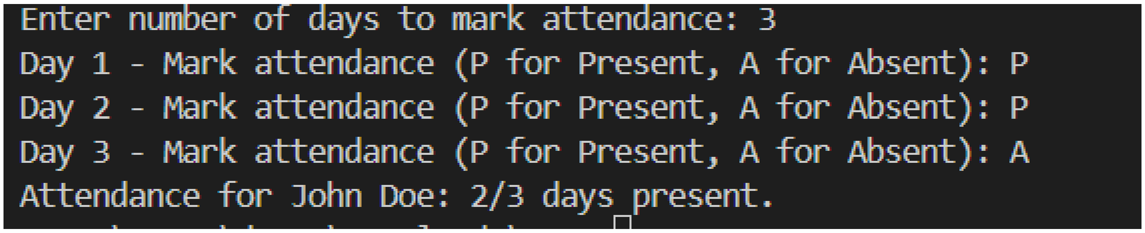
loops.

Code:



```
#task4:
#Generate a Python class to mark and display student attendance using loops.
class Student:
    def __init__(self, name):
        self.name = name
        self.attendance = []
    def mark_attendance(self, status):
        self.attendance.append(status)
    def display_attendance(self):
        present_days = self.attendance.count('P')
        total_days = len(self.attendance)
        print(f"Attendance for {self.name}: {present_days}/{total_days} days present.")
student = Student("John Doe")
days = int(input("Enter number of days to mark attendance: "))
for day in range(1, days + 1):
    status = input(f"Day {day} - Mark attendance (P for Present, A for Absent): ").upper()
    while status not in ['P', 'A']:
        print("Invalid input. Please enter 'P' for Present or 'A' for Absent.")
        status = input(f"Day {day} - Mark attendance (P for Present, A for Absent): ").upper()
    student.mark_attendance(status)
student.display_attendance()
```

Result:



```
Enter number of days to mark attendance: 3
Day 1 - Mark attendance (P for Present, A for Absent): P
Day 2 - Mark attendance (P for Present, A for Absent): P
Day 3 - Mark attendance (P for Present, A for Absent): A
Attendance for John Doe: 2/3 days present.
```

Observation

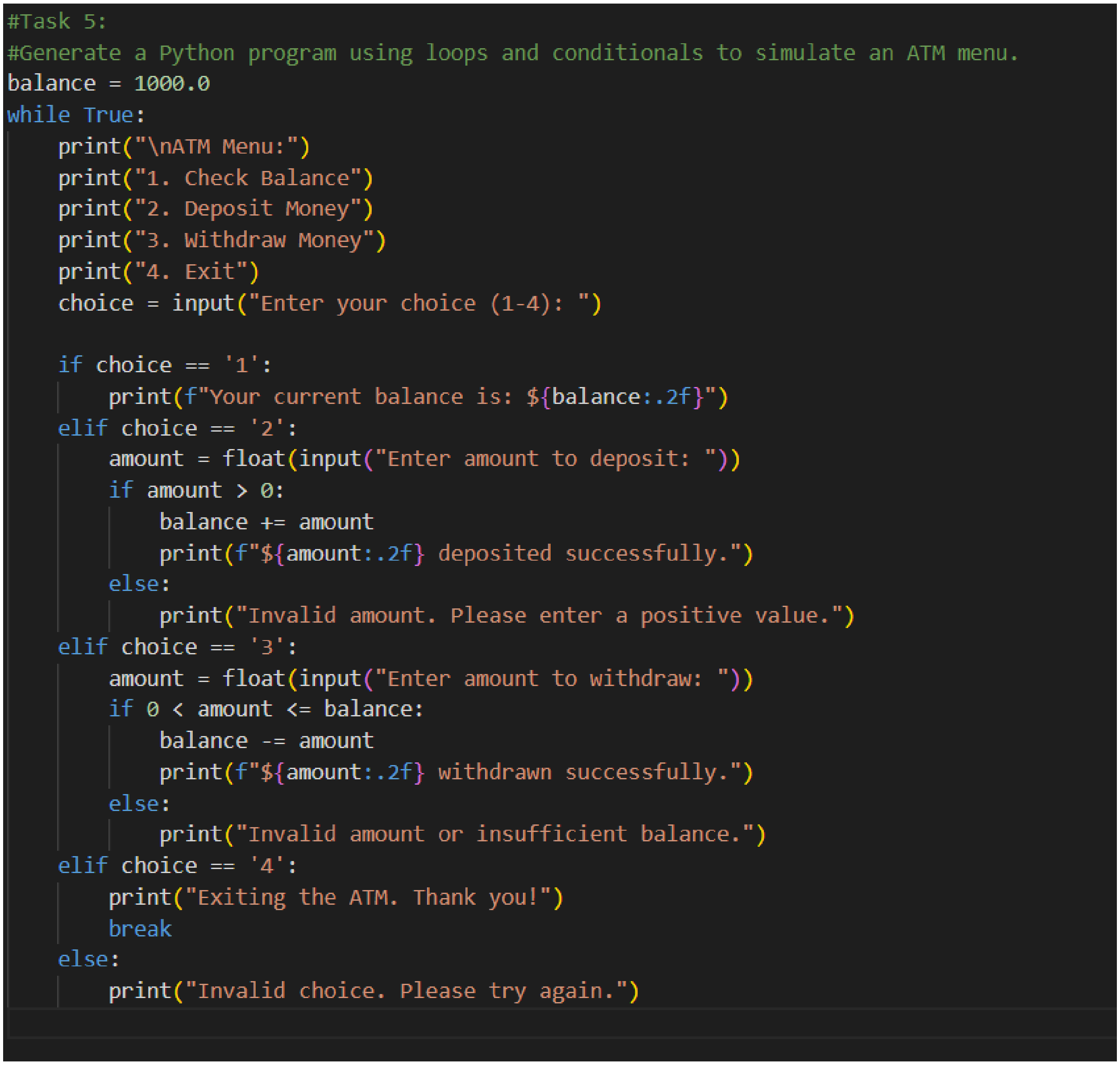
The AI-generated student attendance system uses an object-oriented approach to record and track attendance for a student. It allows attendance to be marked day by day with basic input validation and accurately calculates the number of days present out of the total days. However, the system is limited to a single student, does not store attendance records persistently, and lacks support for attendance analysis such as percentages or multiple students, reducing its applicability in real-world academic environments.

Task 5: AI-Based Code Completion for Conditional Menu Navigation

The purpose of this task is use an AI tool to complete a navigation menu.

Prompt: # Generate a Python program using loops and conditionals to simulate an ATM menu.

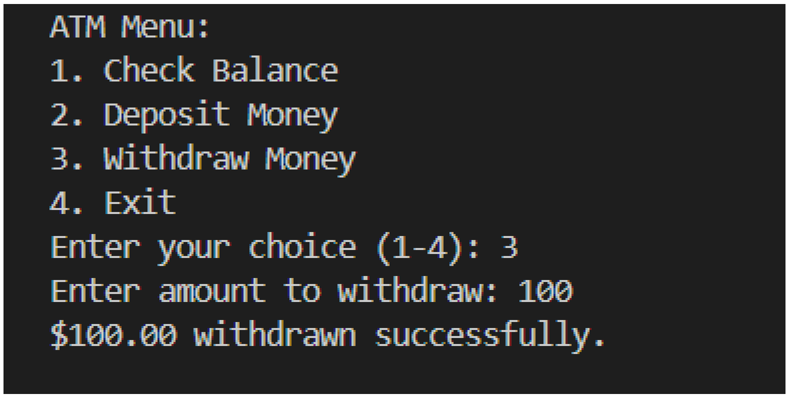
Code:



```
#Task 5:
#Generate a Python program using loops and conditionals to simulate an ATM menu.
balance = 1000.0
while True:
    print("\nATM Menu:")
    print("1. Check Balance")
    print("2. Deposit Money")
    print("3. Withdraw Money")
    print("4. Exit")
    choice = input("Enter your choice (1-4): ")

    if choice == '1':
        print(f"Your current balance is: ${balance:.2f}")
    elif choice == '2':
        amount = float(input("Enter amount to deposit: "))
        if amount > 0:
            balance += amount
            print(f"${amount:.2f} deposited successfully.")
        else:
            print("Invalid amount. Please enter a positive value.")
    elif choice == '3':
        amount = float(input("Enter amount to withdraw: "))
        if 0 < amount <= balance:
            balance -= amount
            print(f"${amount:.2f} withdrawn successfully.")
        else:
            print("Invalid amount or insufficient balance.")
    elif choice == '4':
        print("Exiting the ATM. Thank you!")
        break
    else:
        print("Invalid choice. Please try again.")
```

Result:



```
ATM Menu:
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Enter your choice (1-4): 3
Enter amount to withdraw: 100
$100.00 withdrawn successfully.
```

Observation

The AI-generated ATM simulation program provides basic banking operations such as checking balance, depositing money, and withdrawing money through a menu-driven interface. It correctly updates the account balance and includes simple validation for deposit and withdrawal amounts. However, the system lacks user authentication, exception handling for invalid numeric inputs, transaction history, and data persistence, making it unsuitable for real-world banking applications.