Lab-6.5

Course : AI Assisted Coding

Topic:- AI-Based Code Completion: Working with suggestions for classes, loops, conditionals

---

Student Details:

Name : Mohammed Sabir

Hall Ticket No : 2303A51506

Batch : 22

Task-01

Final Optimal Prompt:-

generate a python ccode to cccheck the voying eligibility based on age and citizenship the code should gave user defined input

if my age is less then 0 it says age can not be less then zero if my citizen ship is other then india it says your citizen ship should be india if i gave age in string and citizenship in numbers it handle the errors but seperately

example 1: input age :sdf output : please enter a valid age

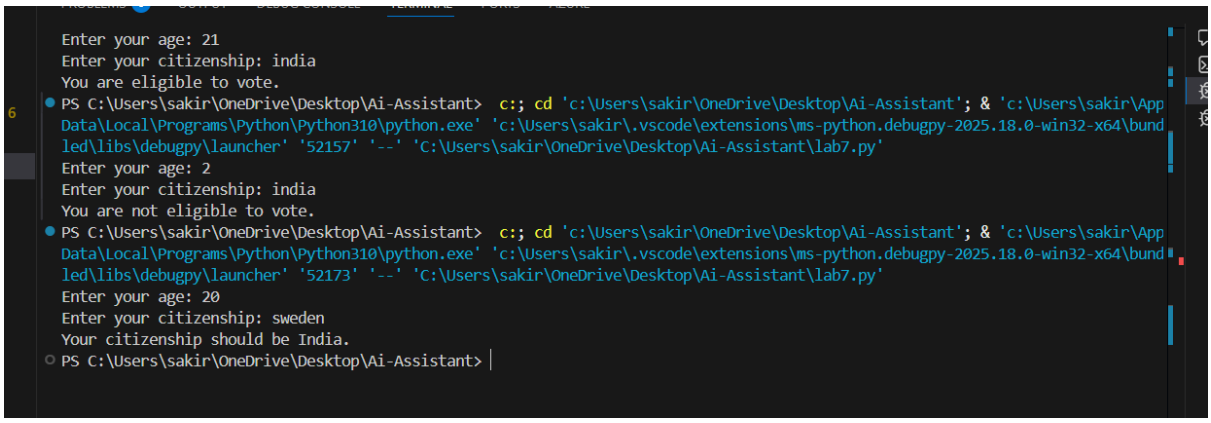
example 2:input citizenship :123 output : please enter a valid citizenship

---

Code Screenshot :

```python
try:
    age_input = input("Enter your age: ")
    age = int(age_input)
    if age < 0:
        print("Age cannot be less than zero.")
    else:
        citizenship = input("Enter your citizenship: ")
        if not citizenship.isalpha():
            print("Please enter a valid citizenship.")
        elif citizenship.lower() != "india":
            print("Your citizenship should be India.")
        else:
            if age >= 18:
                print("You are eligible to vote.")
            else:
                print("You are not eligible to vote.")
except ValueError:
    print("Please enter a valid age.")
```

Output Screenshot:-



Explanation/Justification/Observation (100 words / 5 – 6 sentence)

This program uses nested conditional logic and exception handling to determine voter eligibility. It first attempts to convert the age input into an integer; if the user enters text instead of a number, the try-except block catches the ValueError and displays a friendly error message. If the age is valid and non-negative, the code then checks the user's citizenship. It ensures the input is alphabetic using .isalpha() and specifically requires "India" (case-insensitive) for eligibility. Only when both criteria—being at least 18 years old and an Indian citizen—are met, does it confirm the user can vote.

Task-02

Final Optimal Prompt:

generate a python code to count the number of vowels and consonants in a given string input by user

if i gave numbers or special characters it should handle the error and give output as please enter a valid string

example 1: input : hello output : vowels:2 consonants:3

example 2:input :123sabir output : vowels:2 consonants:3 neglect the number and gave answer only for string part

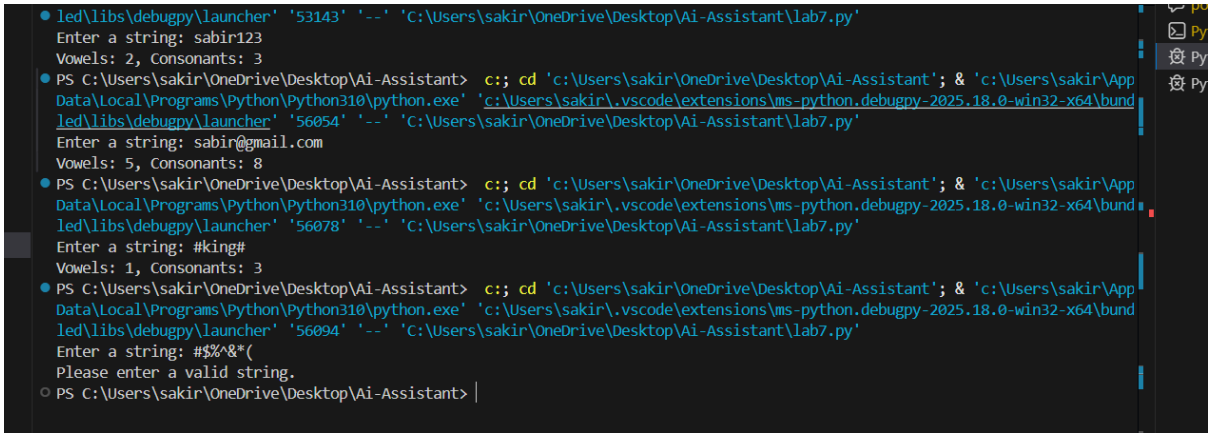
example 3: input : @#$$% output : please enter a valid string

---

Code Screenshot :-

```
33
34  def count_vowels_consonants(s):
35      vowels = "aeiouAEIOU"
36      vowel_count = 0
37      consonant_count = 0
38      has_alpha = False
39      for char in s:
40          if char.isalpha():
41              has_alpha = True
42              if char in vowels:
43                  vowel_count += 1
44              else:
45                  consonant_count += 1
46      if not has_alpha:
47          print("Please enter a valid string.")
48      else:
49          print(f"Vowels: {vowel_count}, Consonants: {consonant_count}")
50  user_input = input("Enter a string: ")
51  count_vowels_consonants(user_input)
52
53
```

Output Screenshot:-



Explanation/Justification/Observation (100 words / 5 – 6 sentence) :-

This program analyzes a string to count vowels and consonants while ensuring the input is valid. It begins by defining a reference string of vowels and initializing counters to zero, alongside a Boolean flag called has_alpha set to False. As it iterates through each character, it uses the .isalpha() method to ignore numbers and symbols, switching the flag to True if at least one letter is found. Inside this check, it uses a membership test to see if the letter is in the vowel string; if so, it increments the vowel count, otherwise, it updates the consonant count. Finally, it uses an if-else structure to either display the results or warn the user if no alphabetic characters were provided.

Task-03

Final Optimal Prompt:

Generate a Python program for a library management system using classes, loops, and conditional statements.

the code should have the following features:

1. add a book to the library

2. remove a book from the library

3. search a book from the library

4. display all the books in the library

5. exit the program

Code Screenshot :-

```
63      class Library:
64          def __init__(self):
65              self.books = []
66          def add_book(self, book):
67              self.books.append(book)
68          def remove_book(self, book):
69              self.books.remove(book)
70          def search_book(self, book):
71              if book in self.books:
72                  return True
73              else:
74                  return False
75          def display_books(self):
76              return self.books
77          def exit_program(self):
78              return "Exiting the program..."
79      if __name__ == "__main__":
80          library = Library()
81          while True:
82              print("1. Add a book")
83              print("2. Remove a book")
84              print("3. Search a book")
85              print("4. Display all the books")
86              print("5. Exit the program")
87              choice = input("Enter your choice: ")
88              if choice == "1":
89                  book = input("Enter the book name: ")
90                  library.add_book(book)
91                  print(f"Book {book} added to the library.")
92              elif choice == "2":
93                  book = input("Enter the book name: ")
94                  library.remove_book(book)
95                  print(f"Book {book} removed from the library.")
96              elif choice == "3":
97                  book = input("Enter the book name: ")
98                  if library.search_book(book):
99                      print(f"Book {book} found in the library.")
100                 else:
101                     print(f"Book {book} not found in the library.")
102             elif choice == "4":
103                 print("All the books in the library:")
104                 for book in library.display_books():
105                     print(book)
106             elif choice == "5":
107                 print(library.exit_program())
108                 break
109             else:
110                 print("Invalid choice. Please enter a valid choice.")
111
```

Output Screenshot:-

```
1. Add a book
2. Remove a book
3. Search a book
4. Display all the books
5. Exit the program
Enter your choice: 1
Enter the book name: one piece
Book one piece added to the library.
1. Add a book
2. Remove a book
3. Search a book
4. Display all the books
5. Exit the program
Enter your choice: 1
Enter the book name: king of arabia
Book king of arabia added to the library.
1. Add a book
2. Remove a book
3. Search a book
4. Display all the books
5. Exit the program
Enter your choice: 4
All the books in the library:
one piece
king of arabia
1. Add a book
2. Remove a book
3. Search a book
4. Display all the books
5. Exit the program
Enter your choice: 3
Enter the book name: king of arabia
Book king of arabia found in the library.
1. Add a book
2. Remove a book
3. Search a book
4. Display all the books
5. Exit the program
```

Explanation/Justification/Observation (100 words / 5 – 6 sentence) :-

This code demonstrates **Object-Oriented Programming (OOP)** by creating a Library class to manage a collection of books. The class acts as a blueprint, using the __init__ method to

initialize an empty list, self.books, which stores the book titles. Each method inside the class—add_book, remove_book, and search_book—performs a specific action on that list, such as appending a new title or checking for its existence using a membership test.

The main part of the script runs a **while loop**, creating an interactive menu that allows users to pick an action. Based on the user's input (1–5), the program calls the corresponding class method to update or view the library's data. This structure keeps the logic organized, separating the data management (the Class) from the user interface (the Menu).

Task-04

Final Optimal Prompt:-

generate a python code mark attendence in management class using python class to mark and dispaly student attendence using loops

the code should have the following features:

1. add a student to the class

2. mark attendence for a student

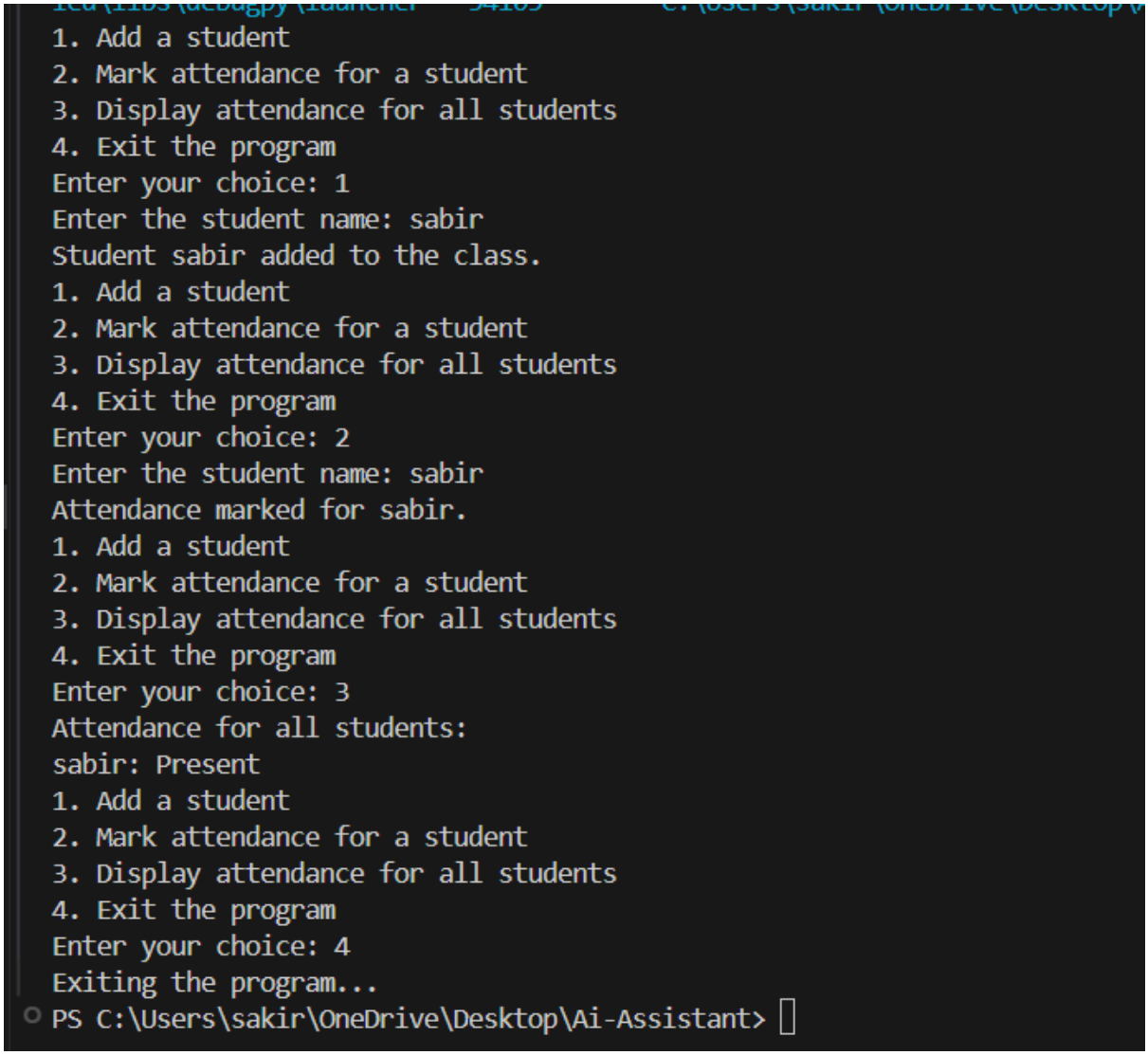
3. display attendence for all students

Code Screenshot :-

```python
class Attendance:
    def __init__(self):
        self.students = {}
    def add_student(self, student):
        self.students[student] = False
    def mark_attendance(self, student):
        if student in self.students:
            self.students[student] = True
        else:
            print(f"Student {student} not found in the class.")
    def display_attendance(self):
        for student, attended in self.students.items():
            status = "Present" if attended else "Absent"
            print(f"{student}: {status}")
if __name__ == "__main__":
    attendance = Attendance()
    while True:
        print("1. Add a student")
        print("2. Mark attendance for a student")
        print("3. Display attendance for all students")
        print("4. Exit the program")
        choice = input("Enter your choice: ")
        if choice == "1":
            student = input("Enter the student name: ")
            attendance.add_student(student)
            print(f"Student {student} added to the class.")
        elif choice == "2":
            student = input("Enter the student name: ")
            attendance.mark_attendance(student)
            print(f"Attendance marked for {student}.")
        elif choice == "3":
            print("Attendance for all students:")
            attendance.display_attendance()
        elif choice == "4":
            print("Exiting the program...")
            break
        else:
            print("Invalid choice. Please enter a valid choice.")

```

Output Screenshot:-



Explanation/Justification/Observation (100 words / 5 – 6 sentence) :-

This program uses Object-Oriented Programming to manage student records using a dictionary for efficient lookups. The Attendance class stores student names as keys and their presence as Boolean values, defaulting to False (Absent) upon registration. The mark_attendance method updates these values to True only if the student exists in the records. When displaying results, the code iterates through the dictionary, translating Booleans into "Present" or "Absent" labels. This design ensures that attendance is tracked individually and accurately through an interactive menu, separating data management logic from the user interface.

Tack-05

Final Optimal Prompt:-

generate a python code for stimulaion of an ATM Menu Using loops and conditional statements the code should have the following features:

1. check balance

2. withdraw money

3. deposit money

4. exit the program

user should give balance amount as input and the code should handle the errors if user give balance in string or negative number

---

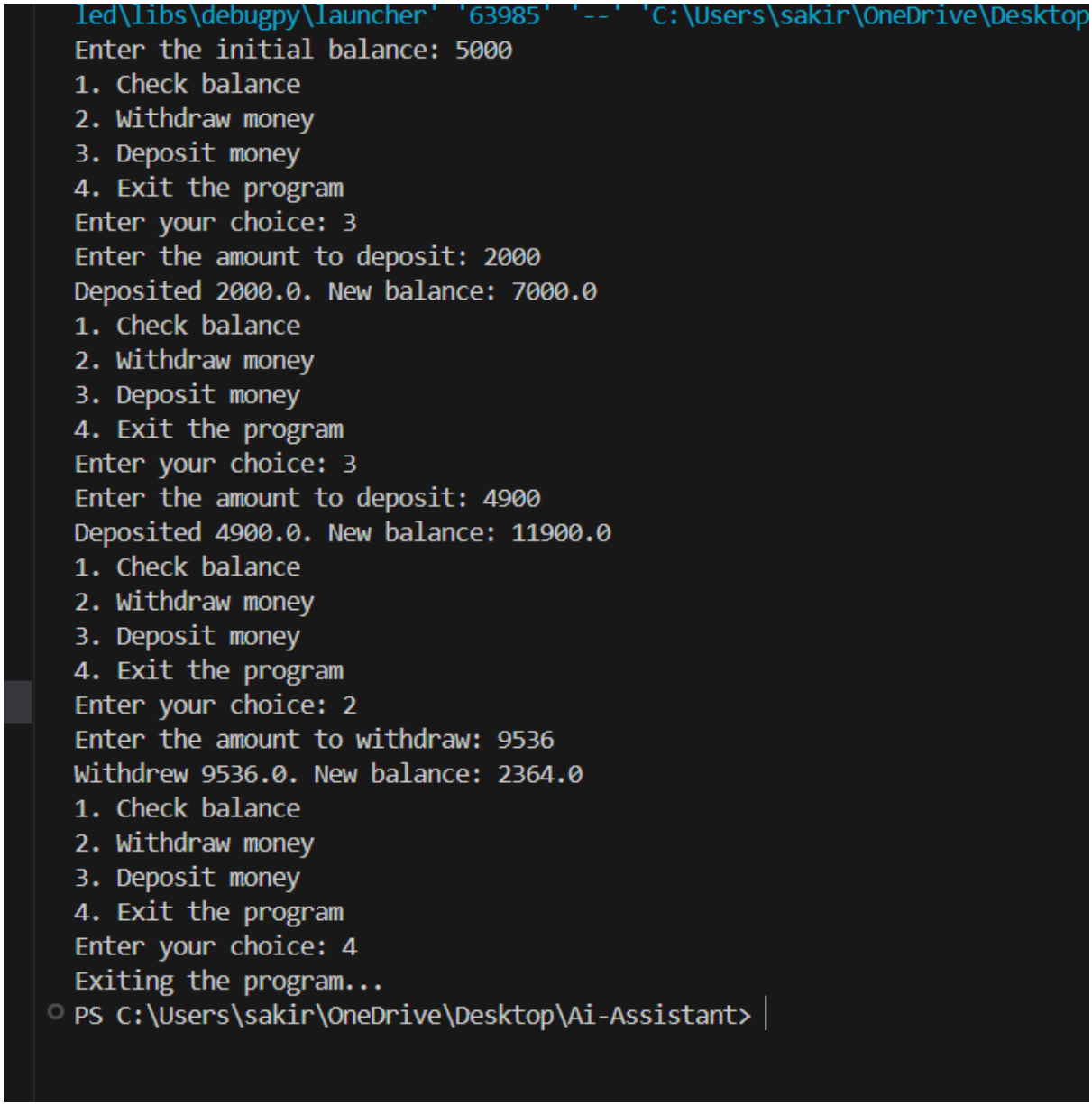
Code Screenshot :-

```python
class ATM:
    def __init__(self, initial_balance=0):
        self.balance = initial_balance
    def check_balance(self):
        return self.balance
    def withdraw_money(self, amount):
        if amount > self.balance:
            print("Insufficient balance.")
        else:
            self.balance -= amount
            print(f"Withdrew {amount}. New balance: {self.balance}")
    def deposit_money(self, amount):
        self.balance += amount
        print(f"Deposited {amount}. New balance: {self.balance}")

if __name__ == "__main__":
    while True:
        try:
            initial_balance = float(input("Enter the initial balance: "))
            if initial_balance < 0:
                print("Initial balance cannot be negative.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a valid number.")

    atm = ATM(initial_balance)
    while True:
        print("1. Check balance")
        print("2. Withdraw money")
        print("3. Deposit money")
        print("4. Exit the program")
        choice = input("Enter your choice: ")
        if choice == "1":
            print(f"Your balance is: {atm.check_balance()}")
        elif choice == "2":
            try:
                amount = float(input("Enter the amount to withdraw: "))
                atm.withdraw_money(amount)
            except ValueError:
                print("Invalid input. Please enter a valid number.")
        elif choice == "3":
            try:
                amount = float(input("Enter the amount to deposit: "))
                atm.deposit_money(amount)
            except ValueError:
                print("Invalid input. Please enter a valid number.")
        elif choice == "4":
            print("Exiting the program...")
            break
        else:
            print("Invalid choice. Please enter a valid choice.")
```

Output Screenshot:-



Explanation/Justification/Observation (100 words / 5 – 6 sentence) :-

This program uses Object-Oriented Programming (OOP) to simulate a banking system. The ATM class encapsulates a single state—the user's balance—and provides methods to modify it safely. The withdraw_money method includes a critical logical check to prevent the balance from dropping below zero, while deposit_money handles additions.

The script stands out for its robust input validation; it uses try-except blocks and while loops to ensure the user provides valid numbers, preventing crashes from typos. By separating the financial math within the class from the menu interface, the code stays organized and professional.
