Assignment-6.5
AI Assisted Coding

Name:E.Tharun
Ht.no:2303A52352
Batch:36

Task Description #1 (AI-Based Code Completion for Conditional Eligibility Check) **bold text**

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

```
def check_voting_eligibility(age, is_citizen):
    if age >= 18 and is_citizen:
        return "Eligible to vote"
    elif age < 18 and is_citizen:
        return "Not eligible to vote (too young)"
    elif age >= 18 and not is_citizen:
        return "Not eligible to vote (not a citizen)"
    else:
        return "Not eligible to vote (too young and not a citizen)"

# Test cases
print(f"Age 20, Citizen: {check_voting_eligibility(20, True)}")
print(f"Age 16, Citizen: {check_voting_eligibility(16, True)}")
print(f"Age 30, Not Citizen: {check_voting_eligibility(30, False)}")
print(f"Age 17, Not Citizen: {check_voting_eligibility(17, False)}")
```

```
Age 20, Citizen: Eligible to vote
Age 16, Citizen: Not eligible to vote (too young)
Age 30, Not Citizen: Not eligible to vote (not a citizen)
Age 17, Not Citizen: Not eligible to vote (too young and not a citizen)
```

Explanation of Voting Eligibility Conditions:

The Python code defines a function `check_voting_eligibility` that takes two arguments: `age` (an integer) and `is_citizen` (a boolean).

1. **Eligible to vote:** A person is eligible if their `age` is 18 or greater (`age >= 18`) AND they are a citizen (`is_citizen` is `True`).

2. **Not eligible (too young):** If the person is a citizen (`is_citizen` is `True`) but their `age` is less than 18 (`age < 18`), they are not eligible because they are too young.
3. **Not eligible (not a citizen):** If the person is 18 or older (`age >= 18`) but they are not a citizen (`is_citizen` is `False`), they are not eligible.
4. **Not eligible (too young and not a citizen):** In all other cases (i.e., if they are both too young and not a citizen), they are also not eligible.

Task Description #2(AI-Based Code Completion for Loop-Based String Processing) **bold text**

PROMPT: Generate Python code to count vowels and consonants in a string using a loop.

```
def count_vowels_consonants(input_string):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in input_string:
        if char.isalpha():  # Check if the character is an alphabet
            if char in vowels:
                vowel_count += 1
            else:
                    consonant_count += 1

        return vowel_count, consonant_count

    # Test cases
    string1 = "Hello World"
    v1, c1 = count_vowels_consonants(string1)
    print(f"String: '{string1}'\nVowels: {v1}, Consonants: {c1}\n")

    string2 = "Python Programming"
    v2, c2 = count_vowels_consonants(string2)
    print(f"String: '{string2}'\nVowels: {v2}, Consonants: {c2}\n")

    string3 = "AI is amazing!"
    v3, c3 = count_vowels_consonants(string3)
    print(f"String: '{string3}'\nVowels: {v3}, Consonants: {c3}\n")

    string4 = "12345"
    v4, c4 = count_vowels_consonants(string4)
    print(f"String: '{string4}'\nVowels: {v4}, Consonants: {c4}\n")
```

```
String: 'Hello World'
Vowels: 3, Consonants: 7

String: 'Python Programming'
Vowels: 4, Consonants: 13

String: 'AI is amazing!'
Vowels: 6, Consonants: 5

String: '12345'
Vowels: 0, Consonants: 0
```

Explanation of Vowel and Consonant Counter:

The Python code defines a function `count_vowels_consonants` that takes one argument: `input_string`.

1. **Initialization:**
    - `vowels` is a string containing all lowercase and uppercase vowels for quick lookup.
    - `vowel_count` and `consonant_count` are initialized to 0.
2. **Iteration:**
    - The code iterates through each `char` in the `input_string` using a `for` loop.
3. **Character Check:**
    - `if char.isalpha():` ensures that only alphabetic characters are considered, ignoring numbers, spaces, and special symbols.
4. **Counting:**
    - If a character is an alphabet, `if char in vowels:` checks if it's a vowel. If true, `vowel_count` is incremented.
    - Otherwise (if it's an alphabet but not a vowel), it's considered a consonant, and `consonant_count` is incremented.
5. **Return Value:**
    - The function returns a tuple containing `vowel_count` and `consonant_count`.

Output Verification:

- **'Hello World'**: (Vowels: 3 - e, o, o; Consonants: 7 - H, l, l, W, r, l, d)
- **'Python Programming'**: (Vowels: 5 - o, o, a, i; Consonants: 12 - P, y, t, h, n, P, r, g, r, m, m, n, g)
- **'AI is amazing!'**: (Vowels: 6 - A, I, i, a, i; Consonants: 4 - s, m, z, n, g)
- **'12345'**: (Vowels: 0, Consonants: 0) - because no alphabetic characters are present.

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

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

```
class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_borrowed = False

    def __str__(self):
        status = "(Borrowed)" if self.is_borrowed else "(Available)"
        return f"{self.title} by {self.author} (ISBN: {self.isbn}) {status}"

class Library:
    def __init__(self, name):
        self.name = name
        self.books = []

    def add_book(self, book):
        # Check for duplicate ISBN
        if any(b.isbn == book.isbn for b in self.books):
            print(f"Error: A book with ISBN {book.isbn} already exists.")
        else:
            self.books.append(book)
            print(f"'{book.title}' added to the library.")

    def list_all_books(self):
        if not self.books:
            print("The library currently has no books.")
            return
        print(f"\n--- Books in {self.name} ---")
        for book in self.books:
            print(book)
        print("----------------------------------------------------")

    def find_book(self, search_term):
        found_books = [book for book in self.books
                       if search_term.lower() in book.title.lower() or
                       search_term.lower() in book.author.lower() or
                       search_term == book.isbn]
        return found_books

    def borrow_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                if not book.is_borrowed:
                    book.is_borrowed = True
                    print(f"'{book.title}' has been borrowed.")
                    return True
                else:
                    print(f"'{book.title}' is already borrowed.")
                return False
        print(f"Book with ISBN {isbn} not found.")
        return False

    def return_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                if book.is_borrowed:
                    book.is_borrowed = False
                    print(f"'{book.title}' has been returned.")
                    return True
                else:
                    print(f"'{book.title}' was not borrowed.")
                    return False
        print(f"Book with ISBN {isbn} not found.")
        return False

def main():
    my_library = Library("Colab Central Library")

    while True:
        print("\n--- Library Menu ---")
        print("1. Add a new book")
        print("2. List all books")
        print("3. Search for a book")
        print("4. Borrow a book")
        print("5. Return a book")
        print("6. Exit")
        choice = input("Enter your choice: ")

        if choice == '1':
            title = input("Enter title: ")
            author = input("Enter author: ")
            isbn = input("Enter ISBN: ")
            new_book = Book(title, author, isbn)
            my_library.add_book(new_book)
        elif choice == '2':
            my_library.list_all_books()
        elif choice == '3':
            search_term = input("Enter title, author, or ISBN to search: ")
            results = my_library.find_book(search_term)
            if results:
                print("\n--- Search Results ---")
                for book in results:
                    print(book)
                print("----------------------")
            else:
                print("No books found matching your search term.")
        elif choice == '4':
            isbn = input("Enter ISBN of the book to borrow: ")
            my_library.borrow_book(isbn)
        elif choice == '5':
            isbn = input("Enter ISBN of the book to return: ")
            my_library.return_book(isbn)
        elif choice == '6':
            print("Exiting Library Management System. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```

```
--- Library Menu ---
```

```
1. Add a new book
2. List all books
3. Search for a book
4. Borrow a book
5. Return a book
6. Exit
Enter your choice: 6
Exiting Library Management System. Goodbye!
```

Task Description #4 (AI-Assisted Code Completion for Class- Based Aflendance System) **bold text**

PROMPT: Generate a Python class to mark and display student aflendance using loops.

```
class AttendanceSystem:
    def __init__(self, students):
        self.students = students  # List of student names
        self.attendance = {student: [] for student in students} # Dictionary to s

    def mark_attendance(self, date, present_students):
        print(f"\n--- Marking Attendance for {date} ---")
        for student in self.students:
            if student in present_students:
                self.attendance[student].append(True) # Mark as present
                print(f"{student}: Present")
            else:
                self.attendance[student].append(False) # Mark as absent
                print(f"{student}: Absent")

    def display_attendance(self):
        print("\n--- Student Attendance Record ---")
        if not self.attendance or not self.students:
            print("No attendance records available.")
            return

        # Prepare header (Dates)
        dates = []
        if self.attendance[self.students[0]]: # Check if there's any attendance r
            # Assuming all students have the same number of attendance records
            for i in range(len(self.attendance[self.students[0]])):
                dates.append(f"Day  {i+1}")

        header = "Student\t" + "\t".join(dates)
        print(header)
        print("-" * len(header))

        for student, records in self.attendance.items():
            status = []
            for record in records:
                status.append("P" if record else "A")
            print(f"{student}\t" + "\t".join(status))
        print("----------------------------------------------------------------------")

# Test Cases
students_list = ["Alice", "Bob", "Charlie"]
attendance_tracker = AttendanceSystem(students_list)

# Day 1 attendance
attendance_tracker.mark_attendance("2023-10-26",  ["Alice",  "Charlie"])

# Day 2 attendance
attendance_tracker.mark_attendance("2023-10-27", ["Bob"])

# Day 3 attendance
```

```
attendance_tracker.mark_attendance("2023-10-28",  ["Alice",  "Bob",  "Charlie"])

# Display final attendance
attendance_tracker.display_attendance()

# Test with no attendance marked yet
print("\n--- Testing with a new system (no attendance yet) ---")
new_students = ["David", "Eve"]
empty_attendance = AttendanceSystem(new_students)
empty attendance.display attendance()
```

```
--- Marking Attendance for 2023-10-26 ---
Alice: Present
Bob: Absent
Charlie: Present

--- Marking Attendance for 2023-10-27 ---
Alice: Absent
Bob: Present
Charlie: Absent

--- Marking Attendance for 2023-10-28 ---
Alice: Present
Bob: Present
Charlie: Present

--- Student Attendance Record ---
Student Day 1   Day 2   Day 3
------------------------------------------
Alice    P       A       F
Bob      A       P       F
Charlie P       A       F
-----------------------------------------------------------

--- Testing with a new system (no attendance yet) ---

--- Student Attendance Record ---
Student
---------
David
Eve
-----------------------------------
```

Explanation of the Aflendance System Class:

The `AttendanceSystem` class is designed to manage student aflendance records. It utilizes a dictionary to store aflendance for each student and loops to process and display this information.

1. **`__init__(self, students)`:**
   - **Purpose:** Initializes the aflendance system with a list of student names.
   - **`self.students`:** Stores the list of student names.
   - **`self.attendance`:** A dictionary where keys are student names and values are lists. Each list will store boolean values (`True` for present, `False` for absent) for each day aflendance is marked. It's initialized with an empty list for each student.
2. **`mark_attendance(self, date, present_students)`:**
   - **Purpose:** Records aflendance for a specific `date`.
   - It iterates through `self.students` using a loop.

- For each student, it checks if their name is in the `present_students` list (which contains names of students present on that day).
- If a student is in `present_students`, `True` is appended to their aflendance list; otherwise, `False` is appended.
- Prints a message indicating whether each student is present or absent for the given date.

3. **`display_attendance(self)`:**
   - **Purpose:** Prints a formafled table showing the aflendance record for all students across all marked days.
   - It first constructs a header with 'Student' and 'Day X' for each recorded aflendance day.
   - It then loops through each `student` and their `records` (the list of `True`/`False` values) in `self.attendance`.
   - For each record, it converts `True` to 'P' (Present) and `False` to 'A' (Absent).
   - Finally, it prints the student's name followed by their 'P'/'A' status for each day, formafled for readability.

Task Description #5 (AI-Based Code Completion for Conditional Menu Navigation) **bold text**

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

```
def atm_program():
    balance = 1000  # Initial balance
    print("Welcome to the ATM!")

    while True:
        print("\n--- ATM Menu ---")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == '1':
            print(f"Your current balance is: ${balance:.2f}")
        elif choice == '2':
            try:
                amount = float(input("Enter amount to deposit: $"))
                if amount > 0:
                    balance += amount
                    print(f"${amount:.2f} deposited successfully. New balance: ${
                else:
                    print("Deposit amount must be positive.")
            except ValueError:
                print("Invalid input. Please enter a number.")
        elif choice == '3':
            try:
                amount = float(input("Enter amount to withdraw: $"))
                if amount > 0:
                    if balance >= amount:
                        balance -= amount
                        print(f"${amount:.2f} withdrawn successfully. New balance
                    else:
                        print("Insufficient  funds.")
                else:
                    print("Withdrawal amount must be positive.")
            except ValueError:
                print("Invalid input. Please enter a number.")
        elif choice == '4':
            print("Thank you for using the ATM. Goodbye!")
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")

# Run the ATM program
atm_program()
```

```
Welcome to the ATM!

--- ATM Menu ---
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 3
Enter amount to withdraw: $3000
Insufficient funds.

--- ATM Menu ---
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 1
Your current balance is: $1000.00

--- ATM Menu ---
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 2
Enter amount to deposit: $1000
$1000.00 deposited successfully. New balance: $2000.00

--- ATM Menu ---
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 4
Thank you for using the ATM. Goodbye!
```

Explanation of the ATM Simulation Program:

The Python program `atm_program()` simulates a basic ATM (Automated Teller Machine) interface using a `while` loop for the menu and `if-elif-else` statements for handling user choices.

1. **Initialization:**
    - `balance = 1000` : Sets an initial account balance of $1000.
2. **Main Menu Loop:**
    - `while True:`: Creates an infinite loop to keep the ATM menu running until the user chooses to exit.
    - The menu options (Check Balance, Deposit, Withdraw, Exit) are printed to the console.
    - `input("Enter your choice (1-4): ")` : Prompts the user to enter their choice.
3. **Option Handling (Conditional Statements):**

  - **Choice '1' (Check Balance):** Prints the current `balance` formafled to two decimal places.
  - **Choice '2' (Deposit):**
    - Prompts for a deposit `amount`.
    - Uses a `try-except` block to handle `ValueError` if the user enters non-numeric input.
    - Checks if the `amount` is positive. If so, it adds the `amount` to the `balance` and displays the new balance.
    - Otherwise, it prints an error message.
  - **Choice '3' (Withdraw):**
    - Prompts for a withdrawal `amount`.
    - Uses a `try-except` block for `ValueError`.
    - Checks if the `amount` is positive.
    - Further checks if there are sufficient `balance` funds. If yes, it subtracts the `amount` from the `balance` and displays the new balance. Otherwise, it prints an "Insufficient funds" message.
  - **Choice '4' (Exit):** Prints a goodbye message and `break`s out of the `while` loop, ending the program.
  - **Invalid Choice:** If any other input is given, it prints an "Invalid choice" message.

Output Verification:

When the program runs, you will see the ATM menu. Here's a walkthrough of expected interactions:

- **Initial Balance Check (Choice 1):**

```
--- ATM Menu ---
...
Enter your choice (1-4): 1
Your current balance is: $1000.00
```

- **Deposit (Choice 2):**

```
--- ATM Menu ---
...
Enter your choice (1-4): 2
Enter amount to deposit: $200
$200.00 deposited successfully. New balance: $1200.00
```

- **Withdrawal (Choice 3):**

```
--- ATM Menu ---
...
Enter your choice (1-4): 3
Enter amount to withdraw: $300
$300.00 withdrawn successfully. New balance: $900.00
```

- **Withdrawal (Insufficient Funds - Choice 3):**

```
--- ATM Menu ---
...
Enter your choice (1-4): 3
Enter amount to withdraw: $1000
Insufficient funds.
```

- **Exit (Choice 4):**

```
--- ATM Menu ---
...
Enter your choice (1-4): 4
Thank you for using the ATM. Goodbye!
```

The program correctly handles valid inputs, updates the balance, and provides appropriate feedback for various scenarios, including invalid amounts and insufficient funds.