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."

```
def is_eligible_to_vote(age, is_citizen, min_age=18):
    """
    Check voting eligibility.
    - age: int or float (non-negative)
    - is_citizen: bool or string (e.g., 'yes', 'no', 'true', 'false')
    - min_age: minimum voting age (default 18)
    Returns True if eligible, False otherwise.
    """
    if not isinstance(age, (int, float)):
        raise TypeError("age must be a number")
    if age < 0:
        raise ValueError("age cannot be negative")

    if isinstance(is_citizen, str):
        is_citizen = is_citizen.strip().lower() in {"yes", "y", "true", "t", "1", "citizen"}
    else:
        is_citizen = bool(is_citizen)

    return age >= min_age and is_citizen

# Example usages
print(is_eligible_to_vote(20, True))    # True
print(is_eligible_to_vote(17, True))    # False
print(is_eligible_to_vote(30, "yes"))   # True
print(is_eligible_to_vote(30, "no"))    # False
```

```
True
False
True
False
```

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.

```
def count_vowels_consonants(s):
    """
    Count vowels and consonants in string s using a loop.
    Non-letter characters are ignored. Case-insensitive.
    Returns a tuple: (vowel_count, consonant_count).
    """
    vowels = set("aeiou")
    v_count = 0
    c_count = 0
    for ch in s.lower():
        if ch.isalpha():
            if ch in vowels:
                v_count += 1
            else:
                c_count += 1
    return v_count, c_count

# Example usage
text = "Hello, World! 123"
v, c = count_vowels_consonants(text)
print(f"Vowels: {v}, Consonants: {c}")
```

```
Vowels: 3, Consonants: 7
```

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."

```
class Book:
    def __init__(self, book_id, title, author):
        self.id = book_id
        self.title = title
        self.author = author
```

```
        self.borrower = None

    @property
    def available(self):
        return self.borrower is None

    def __str__(self):
        status = "Available" if self.available else f"Borrowed by {self.borrower.name}"
        return f"[{self.id}] {self.title} by {self.author} - {status}"


class Member:
    def __init__(self, member_id, name):
        self.id = member_id
        self.name = name
        self.borrowed_books = []

    def __str__(self):
        return f"[{self.id}] {self.name} (Borrowed: {len(self.borrowed_books)})"


class Library:
    def __init__(self):
        self.books = {}
        self.members = {}
        self.next_book_id = 1
        self.next_member_id = 1

    # Book management
    def add_book(self, title, author):
        book = Book(self.next_book_id, title, author)
        self.books[self.next_book_id] = book
        self.next_book_id += 1
        return book

    def remove_book(self, book_id):
        book = self.books.get(book_id)
        if not book:
            return False, "Book not found."
        if not book.available:
            return False, "Book is currently borrowed and cannot be removed."
        del self.books[book_id]
        return True, "Book removed."

    def list_books(self):
        if not self.books:
            print("No books in the library.")
            return
        for book in sorted(self.books.values(), key=lambda b: b.id):
            print(book)

    # Member management
    def register_member(self, name):
        member = Member(self.next_member_id, name)
        self.members[self.next_member_id] = member
        self.next_member_id += 1
        return member

    def list_members(self):
        if not self.members:
            print("No registered members.")
            return
        for member in sorted(self.members.values(), key=lambda m: m.id):
            print(member)

    # Borrow/return
    def borrow_book(self, member_id, book_id):
        member = self.members.get(member_id)
        book = self.books.get(book_id)
        if not member:
            return False, "Member not found."
        if not book:
            return False, "Book not found."
        if not book.available:
            return False, "Book is already borrowed."
        book.borrower = member
        member.borrowed_books.append(book)
        return True, "Book borrowed successfully."

    def return_book(self, member_id, book_id):
        member = self.members.get(member_id)
        book = self.books.get(book_id)
        if not member or not book:
            return False, "Member or book not found."
```

```
        if book.borrower != member:
            return False, "This book was not borrowed by this member."
        book.borrower = None
        try:
            member.borrowed_books.remove(book)
        except ValueError:
            pass
        return True, "Book returned successfully."


# --- Example usage / simple CLI ---
lib = Library()
# seed data
lib.add_book("1984", "George Orwell")
lib.add_book("To Kill a Mockingbird", "Harper Lee")
lib.register_member("Alice")
lib.register_member("Bob")

def menu():
    print("\nLibrary Menu:")
    print("1. List books")
    print("2. Add book")
    print("3. Remove book")
    print("4. List members")
    print("5. Register member")
    print("6. Borrow book")
    print("7. Return book")
    print("8. Exit")

while True:
    menu()
    choice = input("Choose an option (1-8): ").strip()
    if choice == "1":
        lib.list_books()
    elif choice == "2":
        title = input("Book title: ").strip()
        author = input("Author: ").strip()
        b = lib.add_book(title, author)
        print(f"Added: {b}")
    elif choice == "3":
        try:
            bid = int(input("Book ID to remove: ").strip())
        except ValueError:
            print("Invalid book ID.")
            continue
        ok, msg = lib.remove_book(bid)
        print(msg)
    elif choice == "4":
        lib.list_members()
    elif choice == "5":
        name = input("Member name: ").strip()
        m = lib.register_member(name)
        print(f"Registered: {m}")
    elif choice == "6":
        try:
            mid = int(input("Member ID: ").strip())
            bid = int(input("Book ID: ").strip())
        except ValueError:
            print("Invalid ID.")
            continue
        ok, msg = lib.borrow_book(mid, bid)
        print(msg)
    elif choice == "7":
        try:
            mid = int(input("Member ID: ").strip())
            bid = int(input("Book ID: ").strip())
        except ValueError:
            print("Invalid ID.")
            continue
        ok, msg = lib.return_book(mid, bid)
        print(msg)
    elif choice == "8":
        print("Goodbye.")
        break
    else:
        print("Invalid option. Please enter a number between 1 and 8.")
```

```
Library Menu:
1. List books
2. Add book
3. Remove book
4. List members
5. Register member
6. Borrow book
```

```
7. Return book
8. Exit
Added: [3]  by  - Available

Library Menu:
1. List books
2. Add book
3. Remove book
4. List members
5. Register member
6. Borrow book
7. Return book
8. Exit
Invalid option. Please enter a number between 1 and 8.

Library Menu:
1. List books
2. Add book
3. Remove book
4. List members
5. Register member
6. Borrow book
7. Return book
8. Exit
Invalid option. Please enter a number between 1 and 8.

Library Menu:
1. List books
2. Add book
3. Remove book
4. List members
5. Register member
6. Borrow book
7. Return book
8. Exit
Invalid option. Please enter a number between 1 and 8.

Library Menu:
1. List books
2. Add book
3. Remove book
4. List members
5. Register member
6. Borrow book
7. Return book
8. Exit
Invalid option. Please enter a number between 1 and 8.

Library Menu:
```

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."

```
class AttendanceManager:
    """
    Simple attendance manager.
    - students: dict mapping student name -> dict of date -> bool (present)
    - dates: list of recorded dates (order preserved)
    """
    def __init__(self):
        self.students = {}
        self.dates = []

    def register(self, name):
        """Register a student (no-op if already registered)."""
        if name not in self.students:
            self.students[name] = {}

    def register_many(self, names):
        """Register multiple students using a loop."""
        for n in names:
            self.register(n)

    def mark(self, date, present_names):
        """
        Mark attendance for a date.
        present_names: iterable of student names who are present.
        Students not listed will be marked absent for that date.
        """
        if date not in self.dates:
            self.dates.append(date)
        present_set = set(present_names)
        for name in self.students:  # loop through all registered students
            self.students[name][date] = name in present_set
```

```
    def mark_single(self, name, date, present=True):
        """Mark a single student's attendance for a date."""
        if name not in self.students:
            raise KeyError(f"Student '{name}' is not registered.")
        if date not in self.dates:
            self.dates.append(date)
        self.students[name][date] = bool(present)

    def display(self, date=None):
        """Display attendance. If date is None, show full table; else show that date only."""
        if not self.students:
            print("No students registered.")
            return

        if date is None:
            # Header
            header = ["Name"] + self.dates
            print("\t".join(header))
            # Rows
            for name in sorted(self.students):
                row = [name]
                for d in self.dates:
                    row.append("P" if self.students[name].get(d, False) else "A")
                print("\t".join(row))
        else:
            print(f"Attendance for {date}:")
            for name in sorted(self.students):
                status = "Present" if self.students[name].get(date, False) else "Absent"
                print(f"{name}: {status}")

    def summary(self):
        """Print summary: number of presents and percentage per student."""
        total_days = len(self.dates)
        for name in sorted(self.students):
            present_count = sum(1 for v in self.students[name].values() if v)
            pct = (present_count / total_days * 100) if total_days else 0.0
            print(f"{name}: {present_count}/{total_days} ({pct:.1f}%)")


# Example usage
att = AttendanceManager()
att.register_many(["Alice", "Bob", "Charlie"])
att.mark("2026-01-30", ["Alice", "Charlie"])
att.mark("2026-01-31", ["Bob"])
att.display()                # full table
att.display("2026-01-30")    # single date
att.summary()                # summary per student
```

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.”

```
# Simple ATM menu (short and direct)
balance = 1000.0

while True:
    print("\nATM Menu\n1) Check balance\n2) Deposit\n3) Withdraw\n4) Exit")
    sel = input("Choose (1-4): ").strip()
    if sel == "1":
        print(f"Current balance: ${balance:.2f}")
    elif sel == "2":
        amt = input("Amount to deposit: ").strip()
        try:
            a = float(amt)
            if a > 0:
                balance += a
                print(f"Deposited ${a:.2f}")
            else:
                print("Enter a positive amount.")
        except ValueError:
            print("Invalid amount.")
    elif sel == "3":
        amt = input("Amount to withdraw: ").strip()
        try:
            a = float(amt)
            if a <= 0:
                print("Enter a positive amount.")
            elif a > balance:
                print("Insufficient funds.")
```

```
            else:
                balance -= a
                print(f"Withdrew ${a:.2f}")
        except ValueError:
            print("Invalid amount.")
    elif sel == "4":
        print("Exiting. Have a good day.")
        break
    else:
        print("Invalid option, try again.")
```

```
Current balance: $10000.00

ATM Menu
1) Check balance
2) Deposit
3) Withdraw
4) Exit
Exiting. Have a good day.
```

Start coding or generate with AI.