# ASSIGNMENT-6.5

**HT.NO-2303A51908**

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.”

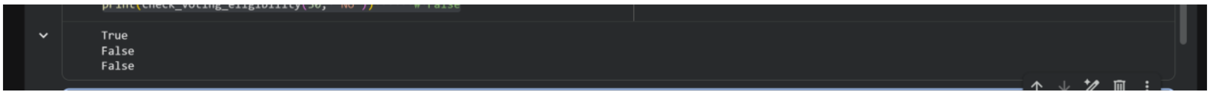
Expected Output:

- AI-generated conditional logic.
- Correct eligibility decisions.
- Explanation of conditions.

**CODE:**

```
def check_voting_eligibility(age: int, citizenship: str) -> bool:
    """
    Returns True if the person is eligible to vote.
    Eligibility: age >= 18 AND citizenship matches required country or 'yes' for citizen flag.
    """
    # Normalize inputs
    citizenship = citizenship.strip().lower()
    # Check age first (fast fail)
    if age < 0:
        raise ValueError("Age cannot be negative.")
    if age < 18:
        return False
    # Check citizenship: allow either exact country name or boolean-like answers
    allowed_yes = {"yes", "y", "true", "citizen"}
    if citizenship in allowed_yes:
        return True
    # If user provided a country name, replace "yourcountry" with the real country you require
    required_country = "india"  # <-- change as needed
    if citizenship == required_country:
        return True
    return False
# Example usages / tests
print(check_voting_eligibility(20, "India"))   # True (if required_country is india)
print(check_voting_eligibility(17, "yes"))     # False
print(check_voting_eligibility(30, "No"))      # False
```

**OUTPUT:**



```
print(check_voting_eligibility(30, "No"))      # False

True
False
False
```

**EXPLANATION:**

- We normalize citizenship to make checks robust.
- We validate age >= 0. AI-generated snippets often forget input normalization and negative-age checks.
- Replace required_country with your country or provide an explicit policy (the assignment likely expects a simple age >= 18 and citizenship == "Yes" approach — above is a slightly more robust version).

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.”

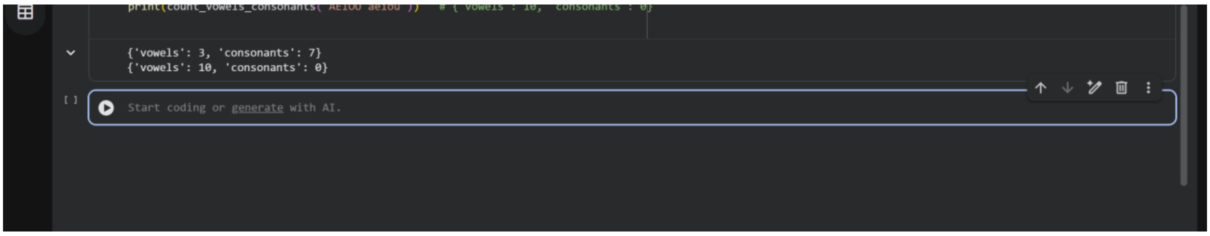
Expected Output:

- AI-generated string processing logic.
- Correct counts.
- Output verification.

**CODE**:

```
def count_vowels_consonants(s: str) -> dict:
    s = s.lower()
    vowels = set("aeiou")
    vowel_count = 0
    consonant_count = 0
    for ch in s:
        if ch.isalpha():
            if ch in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
    return {"vowels": vowel_count, "consonants": consonant_count}
# Example tests
print(count_vowels_consonants("Hello World!"))  # {'vowels': 3, 'consonants': 7}
print(count_vowels_consonants("AEIOU aeiou"))   # {'vowels': 10, 'consonants': 0}
```

**OUTPUT**:



```
print(count_vowels_consonants("AEIOU aeiou"))   # {'vowels': 10, 'consonants': 0}
```

```
{'vowels': 3, 'consonants': 7}
{'vowels': 10, 'consonants': 0}
```

**EXPLANATION**:

- Use str.isalpha() to ignore digits/punctuation — many AI outputs forget this and count punctuation as consonants or letters.
- The loop-based method explicitly shows the algorithm (as required by the assignment). If you want to be concise, a comprehension or sum(...) could be used, but the loop is clearer for demonstration.

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.

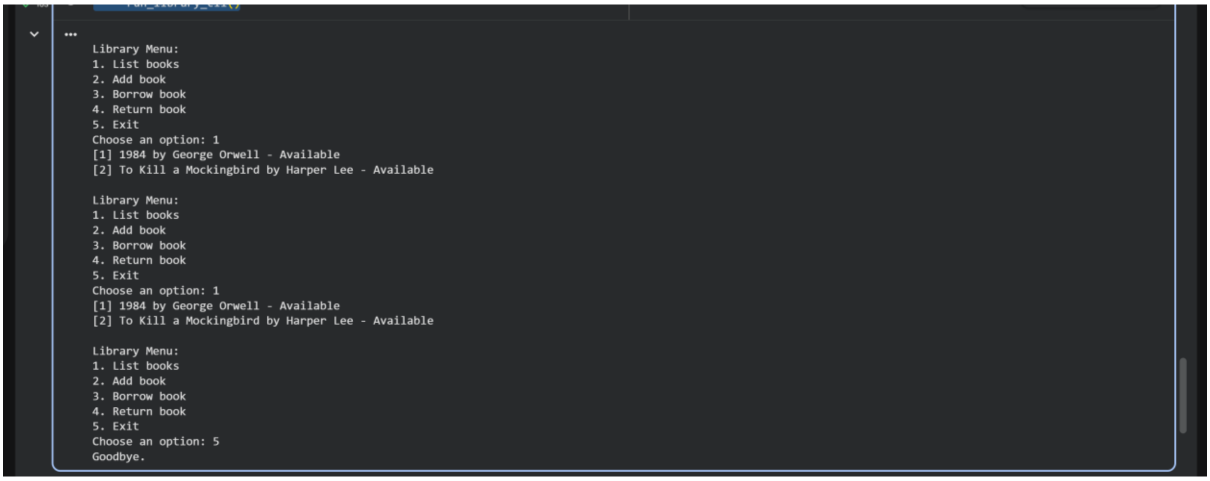
• Review of AI suggestions quality.

• Short reflection on AI-assisted coding experience

**CODE**:

```
class Book:
    def __init__(self, book_id: int, title: str, author: str):
        self.book_id = book_id
        self.title = title
        self.author = author
        self.is_borrowed = False
    def __str__(self):
        status = "Borrowed" if self.is_borrowed else "Available"
        return f"[{self.book_id}] {self.title} by {self.author} - {status}"
class Library:
    def __init__(self):
        self.books = {}  # book_id -> Book
        self.next_id = 1
    def add_book(self, title: str, author: str):
        b = Book(self.next_id, title, author)
        self.books[self.next_id] = b
        self.next_id += 1
        return b.book_id
    def list_books(self):
        return list(self.books.values())
    def borrow_book(self, book_id: int) -> bool:
        book = self.books.get(book_id)
        if not book:
            return False  # not found
        if book.is_borrowed:
            return False  # already borrowed
        book.is_borrowed = True
        return True
    def return_book(self, book_id: int) -> bool:
        book = self.books.get(book_id)
        if not book or not book.is_borrowed:
            return False
        book.is_borrowed = False
        return True
def run_library_cli():
    lib = Library()
    # add some sample books
    lib.add_book("1984", "George Orwell")
    lib.add_book("To Kill a Mockingbird", "Harper Lee")
    while True:
        print("\nLibrary Menu:")
        print("1. List books")
        print("2. Add book")
        print("3. Borrow book")
        print("4. Return book")
        print("5. Exit")
        choice = input("Choose an option: ").strip()
        if choice == "1":
            for b in lib.list_books():
                print(b)
        elif choice == "2":
            title = input("Title: ").strip()
            author = input("Author: ").strip()
            book_id = lib.add_book(title, author)
            print(f"Added with id {book_id}")
        elif choice == "3":
            try:
                bid = int(input("Book id to borrow: ").strip())
            except ValueError:
                print("Invalid id.")
                continue
            ok = lib.borrow_book(bid)
            print("Borrowed." if ok else "Cannot borrow (not found or already borrowed).")
        elif choice == "4":
            try:
                bid = int(input("Book id to return: ").strip())
            except ValueError:
                print("Invalid id.")
                continue
            ok = lib.return_book(bid)
            print("Returned." if ok else "Cannot return (not found or not borrowed).")
        elif choice == "5":
          print("Goodbye.")
            break
        else:
            print("Invalid option. Try again.")
# To run the CLI, uncomment below:
if __name__ == "__main__":
    run_library_cli()
```

**OUTPUT**:



```
Library Menu:
1. List books
2. Add book
3. Borrow book
4. Return book
5. Exit
Choose an option: 1
[1] 1984 by George Orwell - Available
[2] To Kill a Mockingbird by Harper Lee - Available

Library Menu:
1. List books
2. Add book
3. Borrow book
4. Return book
5. Exit
Choose an option: 1
[1] 1984 by George Orwell - Available
[2] To Kill a Mockingbird by Harper Lee - Available

Library Menu:
1. List books
2. Add book
3. Borrow book
4. Return book
5. Exit
Choose an option: 5
Goodbye.
```

**EXPLANATION**:

- The solution uses Book and Library classes, demonstrates loops (menu loop) and conditionals for option handling.
- AI tends to produce huge monolithic programs — prefer small, testable methods (add_book, borrow_book) which are easy to unit test.
- Test ideas: borrow a non-existent book, borrow twice, return without borrowing.

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.

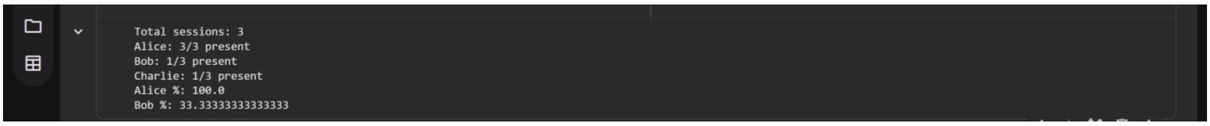
• Test cases

**CODE**:

```
class Attendance:
    def __init__(self, students: list):
        # students: list of student names or ids
        self.students = list(students)
        # Initialize attendance map: student -> list of dates (or boolean per session)
        self.records = {s: [] for s in self.students}

    def mark_attendance(self, date: str, present_list: list):
        """
        date: string like '2026-01-23'
        present_list: list of student names/ids who are present
        """
        present_set = set(present_list)
        for s in self.students:
            self.records[s].append((date, s in present_set))

    def display_attendance(self):
        # Pretty print summary: student -> attendance count / total sessions
        total_sessions = 0
        # infer total sessions from first student (or 0)
        if self.students:
            total_sessions = len(self.records[self.students[0]])
        print(f"Total sessions: {total_sessions}")
        for s in self.students:
            present_count = sum(1 for _, present in self.records[s] if present)
            print(f"{s}: {present_count}/{total_sessions} present")

    def get_attendance_percentage(self, student):
        sessions = self.records.get(student, [])
        if not sessions:
            return 0.0
        present_count = sum(1 for _, present in sessions if present)
        return (present_count / len(sessions)) * 100.0
```

```
# Test cases

students = ["Alice", "Bob", "Charlie"]

att = Attendance(students)

att.mark_attendance("2026-01-20", ["Alice", "Charlie"])

att.mark_attendance("2026-01-21", ["Alice"])

att.mark_attendance("2026-01-22", ["Bob", "Alice"])

att.display_attendance()

print("Alice %:", att.get_attendance_percentage("Alice"))

print("Bob %:", att.get_attendance_percentage("Bob"))
```

**OUTPUT**:



```
Total sessions: 3
Alice: 3/3 present
Bob: 1/3 present
Charlie: 1/3 present
Alice %: 100.0
Bob %: 33.33333333333333
```

**EXPLANATION**:

- records stores tuples (date, present_flag) so you can later expand to reasons, late marks, etc.
- Tests added demonstrate marking multiple sessions and computing percentages.
- AI outputs sometimes use parallel lists incorrectly; prefer dictionary keyed by student for clarity.

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.

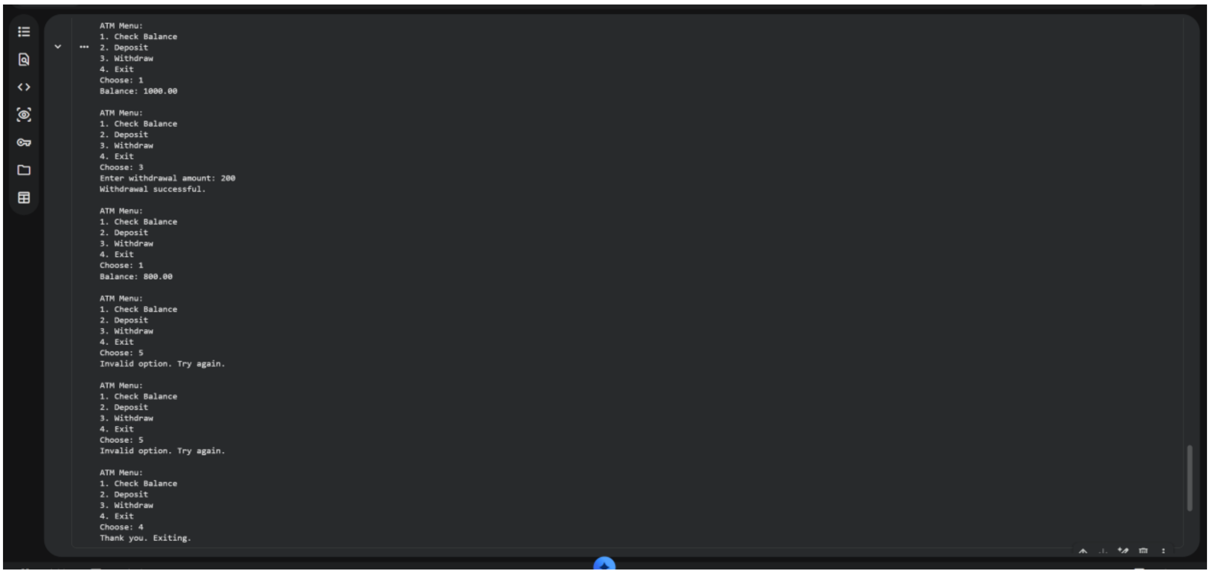
• Correct option handling.

- Output verification

**CODE**:

```
def atm_cli():
    balance = 1000.0  # starting balance
    while True:
        print("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Exit")
        choice = input("Choose: ").strip()
        if choice == "1":
            print(f"Balance: {balance:.2f}")
        elif choice == "2":
            try:
                amt = float(input("Enter deposit amount: ").strip())
                if amt <= 0:
                    print("Enter a positive amount.")
                else:
                    balance += amt
                    print("Deposit successful.")
            except ValueError:
                print("Invalid amount.")
        elif choice == "3":
            try:
                amt = float(input("Enter withdrawal amount: ").strip())
                if amt <= 0:
                    print("Enter a positive amount.")
                elif amt > balance:
                    print("Insufficient funds.")
                else:
                    balance -= amt
                    print("Withdrawal successful.")
            except ValueError:
                print("Invalid amount.")
        elif choice == "4":
            print("Thank you. Exiting.")
            break
        else:
            print("Invalid option. Try again.")
# To run ATM, uncomment:
# if __name__ == "__main__":
#     atm_cli()
```

**OUTPUT**:



```
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 1
Balance: 1000.00

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 3
Enter withdrawal amount: 200
Withdrawal successful.

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 1
Balance: 800.00

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 5
Invalid option. Try again.

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 5
Invalid option. Try again.

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Choose: 4
Thank you. Exiting.
```

**EXPLANATION**:

- This is a standard menu loop with input validation.
- Important AI pitfalls: forgetting to validate numeric input or allowing negative deposits/withdrawals.