

AI -Assistant Coding

Assignment 6.5

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

Name:K.sneha

HT.No:2303A51332

Batch:20

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

Prompt:

Generate a Python program that uses input() to read a person's age and citizenship status (yes/no), applies conditional statements to check voting eligibility, and prints whether the person is eligible to vote.

Code:

```
age_str = input("Please enter the person's age: ")

try:
    age = int(age_str)
except ValueError:
    print("Invalid age. Please enter a number.")

    exit()

# Get citizenship status input from the user

citizen_status_str = input("Is the person a citizen? (yes/no): ").lower()

is_citizen = False

if citizen_status_str == 'yes':
    is_citizen = True

elif citizen_status_str == 'no':
    is_citizen = False

else:
    print("Invalid input for citizenship. Please enter 'yes' or 'no!'")

    exit()
```

```
# Check voting eligibility using conditional statements
```

```
if age >= 18 and is_citizen:
```

```
    print("The person is eligible to vote.")
```

```
else:
```

```
    print("The person is not eligible to vote.")
```

The screenshot shows a Google Colab interface. On the left, a code cell contains Python code for checking voting eligibility. On the right, a Gemini AI panel provides feedback on the code's purpose and execution.

```
# Get age input from the user
age_str = input("Please enter the person's age: ")
try:
    age = int(age_str)
except ValueError:
    print("Invalid age. Please enter a number.")
    exit()

# Get citizenship status input from the user
citizen_status_str = input("Is the person a citizen? (yes/no): ").lower()
is_citizen = False
if citizen_status_str == 'yes':
    is_citizen = True
elif citizen_status_str == 'no':
    is_citizen = False
else:
    print("Invalid input for citizenship. Please enter 'yes' or 'no'.")
    exit()

# Check voting eligibility using conditional statements
if age >= 18 and is_citizen:
    print("The person is eligible to vote.")
else:
    print("The person is not eligible to vote.")
```

Gemini

M Generate a Python program that uses `input()` to read a person's age and citizenship status (yes/no), applies conditional statements to check voting eligibility, and prints whether the person is eligible to vote.

Here's the Python program that takes age and citizenship status as input to determine voting eligibility.

The code executed successfully. Based on the input age of 14 and the citizenship status of 'no', the person is not eligible to vote, which the program correctly determined and printed.

Test the code with an eligible age and citizenship
Improve error handling for invalid citizenship input
Refactor this code into a reusable function

What can I help you build?

Gemini 2.5 Flash ▾>

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

Correct Eligibility Decisions

- If the age is 18 or above and the citizenship input is “yes”, the program prints “Eligible to vote”.
- If either condition fails (age below 18 or not a citizen), the program prints “Not eligible to vote”.
- This ensures only legally eligible individuals are allowed to vote.

Explanation of Conditions

- `age >= 18` checks whether the person meets the minimum legal voting age.
- `citizen.lower() == "yes"` verifies citizenship and converts input to lowercase to avoid case-sensitivity issues.
- The logical AND (`and`) operator ensures both conditions must be true for eligibility.

Observation:

In this task, AI-generated Python code was used to check voting eligibility based on age and citizenship. The conditional logic accurately verified whether the age was 18 or above and whether the individual was a citizen. The program produced correct eligibility decisions for different test inputs, demonstrating proper use of if-else conditions. The logic was simple, readable, and effective for validating eligibility criteria.

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

Prompt:

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

Code:

```
def count_vowels_consonants(input_string):
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0
    for char in input_string:
        if char.isalpha(): # Check if the character is an alphabet
            char_lower = char.lower()
            if char_lower in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
    return vowel_count, consonant_count

# Get input from the user
user_string = input("Enter a string: ")

v, c = count_vowels_consonants(user_string)
print(f"Number of vowels: {v}")
print(f"Number of consonants: {c}")
```

```

[s] ✓ 1ms
def count_vowels_consonants(input_string):
    vowels = "aeiou"
    vowel_count = 0
    consonant_count = 0

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

    return vowel_count, consonant_count

# Get input from the user
user_string = input("Enter a string: ")

v, c = count_vowels_consonants(user_string)

print("Number of vowels: (v)")
print("Number of consonants: (c)")

...
Enter a string: Nandhu
Number of vowels: 2
Number of consonants: 4

```

Correct Counts

- The input string entered is “Nandhu”.
- Vowels present: a, u → Total 2 vowels
- Consonants present: N, n, d, h → Total 4 consonants
- Non-alphabet characters (if any) are ignored using isalpha().

The program correctly identifies and counts vowels and consonants.

Output Verification

Input:

Nandhu

Output:

Number of vowels: 2

Number of consonants: 4

The displayed output matches the expected vowel and consonant counts, confirming that the loop-based string processing logic works correctly.

Observation:

For this task, AI-generated code successfully used loops to process a string and count vowels and consonants. The loop iterated through each character in the string, and alphabetic characters were correctly identified using built-in checks. The output values

matched the expected results, confirming that the loop and condition-based logic worked correctly.

Task 3: AI-Assisted Code Completion Reflection Task (Library Management System)

Prompt:

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

Code:

```
class Book:  
    def __init__(self, title, author, isbn):  
        self.title = title  
        self.author = author  
        self.isbn = isbn  
  
    def __str__(self):  
        return f"Title: {self.title}, Author: {self.author}, ISBN: {self.isbn}"  
  
class Library:  
    def __init__(self):  
        self.books = []  
  
    def add_book(self, title, author, isbn):  
        book = Book(title, author, isbn)  
        self.books.append(book)  
        print(f"Book '{title}' added to the library.")  
  
    def list_all_books(self):  
        if not self.books:  
            print("The library is empty.")  
            return  
  
        print("\n--- All Books in Library ---")  
        for book in self.books:  
            print(book)
```

```
print("-----")  
  
def search_book(self, query):  
    found_books = []  
    query_lower = query.lower()  
  
    for book in self.books:  
        if query_lower in book.title.lower() or query_lower in book.author.lower():  
            found_books.append(book)  
  
    if found_books:  
        print(f"\n--- Search Results for '{query}' ---")  
  
        for book in found_books:  
            print(book)  
  
        print("-----")  
  
    else:  
        print(f"No books found matching '{query}'")  
  
# Main program  
  
def run_library_system():  
    library = Library()  
  
    while True:  
        print("\nLibrary Management System Menu:")  
        print("1. Add a new book")  
        print("2. List all books")  
        print("3. Search for a book")  
        print("4. Exit")  
  
        choice = input("Enter your choice (1-4): ")  
  
        if choice == '1':  
            title = input("Enter book title: ")  
            author = input("Enter book author: ")  
            isbn = input("Enter book ISBN: ")
```

```

library.add_book(title, author, isbn)

elif choice == '2':

    library.list_all_books()

elif choice == '3':

    query = input("Enter title or author to search: ")

    library.search_book(query)

elif choice == '4':

    print("Exiting Library Management System. Goodbye!")

    break

else:

    print("Invalid choice. Please enter a number between 1 and 4.")

# Run the system

run_library_system()

```

The screenshot shows a Google Colab notebook interface. On the left, the code for the library management system is displayed. On the right, a Gemini AI sidebar provides a summary of the program's execution and asks for feature suggestions.

Code (Left):

```

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

    def __str__(self):
        return f"Title: {self.title}, Author: {self.author}, ISBN: {self.isbn}"

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

    def add_book(self, title, author, isbn):
        book = Book(title, author, isbn)
        self.books.append(book)
        print(f"Book '{title}' added to the library.")

    def list_all_books(self):
        if not self.books:
            print("The library is empty.")
            return
        print("\n--- All Books in Library ---")
        for book in self.books:
            print(book)
        print("-----")

    def search_book(self, query):
        found_books = []
        query_lower = query.lower()
        for book in self.books:
            if query_lower in book.title.lower() or query_lower in book.author.lower():
                found_books.append(book)
        return found_books

```

Gemini Summary (Right):

- Generate a Python program for a library management system using classes, loops, and conditional statements.
- Here's a Python program for a simple library management system using classes, loops, and conditional statements. You can add books, list all books, and search for books by title or author.
- The library management system program executed successfully. It demonstrates the full functionality:
 - It displayed the main menu.
 - You successfully added a book titled 'Sudhamurthy's Grandma Stories' by 'Sudhamurthy' with ISBN '123456'.
 - It then listed all books, showing the newly added book.
 - A search for 'Sudhamurthy' correctly returned 'Sudhamurthy's Grandma Stories'.
 - Finally, the system exited gracefully.

AI Interaction:

- Add a feature to the library system
- Explain the vowel counting code
- How to save the library data to a file

What can I help you build?

Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. Learn more

The screenshot shows a Google Colab interface with two main panes. The left pane displays a Python script for a library management system. The right pane shows the AI-generated code and its execution results.

```

if query_lower in book.title.lower() or query_lower in book.author.lower():
    found_books.append(book)

if found_books:
    print(f"\n--- Search Results for '{query}' ---")
    for book in found_books:
        print(book)
    print("-----")
else:
    print(f"No books found matching '{query}'.")

# Main program
def run_library_system():
    library = Library()

    while True:
        print("\nLibrary Management System Menu:")
        print("1. Add a new book")
        print("2. List all books")
        print("3. Search for a book")
        print("4. Exit")

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

        if choice == '1':
            title = input("Enter book title: ")
            author = input("Enter book author: ")
            isbn = input("Enter book ISBN: ")
            library.add_book(title, author, isbn)
        elif choice == '2':
            library.list_all_books()
        elif choice == '3':
            query = input("Enter title or author to search: ")

```

Gemini

- Generate a Python program for a library management system using classes, loops, and conditional statements.
- Here's a Python program for a simple library management system using classes, loops, and conditional statements. You can add books, list all books, and search for books by title or author. The library management system program executed successfully. It demonstrates the full functionality:
 - It displayed the main menu.
 - You successfully added a book titled 'Sudhamurthy's Grandma Stories' by 'Sudhamurthy' with ISBN '123456'.
 - It then listed all books, showing the newly added book.
 - A search for 'Sudhamurthy' correctly returned 'Sudhamurthy's Grandma Stories'.
 - Finally, the system exited gracefully.

The screenshot shows the execution results of the AI-generated Python code. The code adds a book ('Sudhamurthy's Grandma Stories' by Sudhamurthy, ISBN 123456) and lists all books, showing the newly added book.

```

Library Management System Menu:
1. Add a new book
2. List all books
3. Search for a book
4. Exit
Enter your choice (1-4): 1
Enter book title: Sudhamurthy's Grandma Stories
Enter book author: Sudhamurthy
Enter book ISBN: 123456
Book 'Sudhamurthy's Grandma Stories' added to the library.

Library Management System Menu:
1. Add a new book
2. List all books
3. Search for a book
4. Exit
Enter your choice (1-4): 2
--- All Books in Library ---
Title: Sudhamurthy's Grandma Stories, Author: Sudhamurthy, ISBN: 123456
-----
Library Management System Menu:
1. Add a new book
2. List all books
3. Search for a book
4. Exit
Enter your choice (1-4): 3
Enter title or author to search: Sudhamurthy
--- Search Results for 'Sudhamurthy' ---
Title: Sudhamurthy's Grandma Stories, Author: Sudhamurthy, ISBN: 123456
-----
Library Management System Menu:
1. Add a new book

```

Review of AI Suggestions Quality

- The AI-generated code correctly uses classes (Book, Library), fulfilling object-oriented design principles.
- Loops (while True, for) are effectively used to:
 - Display menus repeatedly
 - Traverse the list of books

- Conditional statements (if–elif–else) handle:
 - Menu navigation
 - Book search logic
 - Empty library cases
- The program is fully executable in Google Colab and produces correct outputs for:
 - Adding books
 - Listing all books
 - Searching by title or author
- Code readability is good due to:
 - Meaningful method names
 - Clear menu-driven structure

Limitations identified:

- No input validation for empty strings or duplicate ISBNs
- Data is stored only in memory (no file/database storage)

Short Reflection on AI-Assisted Coding Experience

Using AI-assisted code completion significantly improved development speed by generating a complete and functional program structure. The AI helped in organizing logic using classes, loops, and conditionals efficiently. However, human review was essential to understand the code flow, test edge cases, and identify missing features such as validation and persistence. This experiment shows that AI is a powerful support tool, but responsible usage requires critical thinking and manual refinement.

Observation:

In the library management system task, AI-generated code effectively combined classes, loops, and conditional statements to create a menu-driven program. The system correctly handled adding books, listing all books, and searching for books by title or author. The use of object-oriented design improved code organization and readability, and the menu loop allowed continuous interaction until exit.

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

Prompt:

Generate a Python class to mark and display student attendance using loops and user input

Code:

```
class Student:  
    def __init__(self, student_id, name):  
        self.student_id = student_id  
        self.name = name  
    def __str__(self):  
        return f"ID: {self.student_id}, Name: {self.name}"  
  
class AttendanceSystem:  
    def __init__(self):  
        self.students = []  
        self.attendance_records = {}  
    def add_student(self, student_id, name):  
        if any(s.student_id == student_id for s in self.students):  
            print(f"Student with ID {student_id} already exists.")  
            return  
        student = Student(student_id, name)  
        self.students.append(student)  
        self.attendance_records[student_id] = {}  
        print(f"Student {name} (ID: {student_id}) added.")  
    def mark_attendance(self, date, student_id, status):  
        if student_id not in self.attendance_records:  
            print(f"Student with ID {student_id} not found.")  
            return  
        valid_statuses = ['present', 'absent', 'late']  
        if status.lower() not in valid_statuses:  
            print(f"Invalid status: {status}. Please use one of {valid_statuses}.")
```

```

    return

if date not in self.attendance_records[student_id]:
    self.attendance_records[student_id][date] = status.lower()
    print(f"Attendance for {student_id} on {date} marked as {status.capitalize()}.")

else:
    print(f"Attendance for {student_id} on {date} already marked. Current:
{self.attendance_records[student_id][date].capitalize()}")

def display_attendance_by_date(self, date):
    print(f"\n--- Attendance for {date} ---")
    found_records = False

    if not self.students:
        print("No students in the system.")
        return

    for student in self.students:
        if date in self.attendance_records[student.student_id]:
            status = self.attendance_records[student.student_id][date]
            print(f"{student.name} (ID: {student.student_id}): {status.capitalize()}")
            found_records = True

    if not found_records:
        print(f"No attendance records found for {date}.")
        print("-----")

def display_student_attendance(self, student_id):
    print(f"\n--- Attendance for Student ID: {student_id} ---")
    if student_id in self.attendance_records:
        if self.attendance_records[student_id]:
            for date, status in self.attendance_records[student_id].items():
                print(f"Date: {date}, Status: {status.capitalize()}")
        else:
            print("No attendance records found for this student ID."))

    else:
        print(f"No student found with ID: {student_id}.")

```

```
    print(f"No attendance records for student {student_id}.")  
else:  
    print(f"Student with ID {student_id} not found.")  
    print("-----")  
  
def display_all_attendance_report(self):  
    print("\n--- Full Attendance Report ---")  
    if not self.students:  
        print("No students in the system. Add some students first.")  
        return  
  
    all_dates = set()  
    for student_id in self.attendance_records:  
        all_dates.update(self.attendance_records[student_id].keys())  
    sorted_dates = sorted(list(all_dates))  
  
    if not sorted_dates:  
        print("No attendance records available. Mark attendance for students first.")  
        return  
  
    # Print header with dates  
  
    header = "Student Name (ID)" + "\t" * 2 # Adjust spacing as needed  
    for date in sorted_dates:  
        header += f"{date}\t"  
    print(header)  
    print("-" * len(header) * 2)  
  
    # Print attendance for each student  
  
    for student in self.students:  
        row = f"{student.name} ({student.student_id})\t"  
        for date in sorted_dates:  
            status = self.attendance_records[student.student_id].get(date, 'N/A')  
            row += f"{status.capitalize()}\t"
```

```
    print(row)
    print("-----")
# Main interactive program

def run_attendance_system():

    attendance_system = AttendanceSystem()

while True:

    print("\nAttendance Management System Menu:")

    print("1. Add a new student")

    print("2. Mark attendance")

    print("3. Display attendance by date")

    print("4. Display student's attendance")

    print("5. Display full attendance report")

    print("6. Exit")

choice = input("Enter your choice (1-6): ")

if choice == '1':

    student_id = input("Enter student ID: ")

    name = input("Enter student name: ")

    attendance_system.add_student(student_id, name)

elif choice == '2':

    date = input("Enter date (YYYY-MM-DD): ")

    student_id = input("Enter student ID: ")

    status = input("Enter status (present/absent/late): ")

    attendance_system.mark_attendance(date, student_id, status)

elif choice == '3':

    date = input("Enter date to display attendance for (YYYY-MM-DD): ")

    attendance_system.display_attendance_by_date(date)

elif choice == '4':

    student_id = input("Enter student ID to display attendance for: ")
```

```

attendance_system.display_student_attendance(student_id)

elif choice == '5':
    attendance_system.display_all_attendance_report()

elif choice == '6':
    print("Exiting Attendance Management System. Goodbye!")

    break

else:
    print("Invalid choice. Please enter a number between 1 and 6.")

# Run the system

run_attendance_system()

```

The screenshot shows a Google Colab notebook interface. On the left, there is a code editor containing Python code for an attendance system. The code includes classes for `Student` and `AttendanceSystem`, methods for adding students, marking attendance, and displaying attendance. A callout from the Gemini AI interface suggests generating a Python class to mark and display student attendance using loops and user input. The AI interface also provides a summary of the task and a button to "Accept & Run". The status bar at the bottom indicates it's 1:33 PM, using Python 3, and shows system information like RAM, Disk, and network.

The screenshot shows the same Google Colab interface after the user accepted the Gemini suggestion. The code has been updated to include a new method `display_attendance_by_date` and a modified `display_student_attendance` method. The AI interface is still visible on the right, showing the summary and the "Accept" button.

```

class AttendanceSystem:
    def __init__(self):
        self.attendance_records = {}
        self.students = {}

    def add_student(self, student_id, name):
        self.students[student_id] = name
        print(f"Student {name} (ID: {student_id}) added.")

    def mark_attendance(self, student_id, date, status):
        if student_id in self.attendance_records:
            self.attendance_records[student_id].update({date: status})
        else:
            self.attendance_records[student_id] = {date: status}
        print(f"Attendance marked for {name} on {date} as {status}.")

    def display_attendance_by_date(self, date):
        for student_id, name in self.students.items():
            if date in self.attendance_records.get(student_id, {}):
                print(f"{name} ({student_id}): {self.attendance_records[student_id].get(date)}")
            else:
                print(f"{name} ({student_id}): N/A")

    def display_full_attendance_report(self):
        header = "Student Name (ID) " + (" " * 2) # Adjust spacing as needed
        for student_id, name in self.students.items():
            header += f"({student_id}) "
        print(header)
        print("-" * len(header))

    def main(self):
        while True:
            choice = input("Enter your choice (1-6): ")
            if choice == '1':
                student_id = input("Enter student ID: ")
                name = input("Enter student name: ")
                self.add_student(student_id, name)
            elif choice == '2':
                date = input("Enter date (YYYY-MM-DD): ")
                student_id = input("Enter student ID: ")
                status = input("Enter status (present/absent/late): ")
                self.mark_attendance(student_id, date, status)
            elif choice == '3':
                date = input("Enter date to display attendance for (YYYY-MM-DD): ")
                self.display_attendance_by_date(date)
            elif choice == '4':
                student_id = input("Enter student ID to display attendance for: ")
                self.display_attendance_by_date(student_id)
            elif choice == '5':
                self.display_full_attendance_report()
            elif choice == '6':
                print("Exiting Attendance Management System. Goodbye!")
                break
            else:
                print("Invalid choice. Please enter a number between 1 and 6.")


attendance_system = AttendanceSystem()
attendance_system.main()

```

```

print("Attendance Management System Menu:")
print("1. Add a new student")
print("2. Mark attendance")
print("3. Display attendance by date")
print("4. Display student's attendance")
print("5. Display full attendance report")
print("6. Exit")

choice = input("Enter your choice (1-6): ")

if choice == '1':
    student_id = input("Enter student ID: ")
    name = input("Enter student name: ")
    attendance_system.add_student(student_id, name)
elif choice == '2':
    date = input("Enter date (YYYY-MM-DD): ")
    student_id = input("Enter student ID: ")
    status = input("Enter status (present/absent/late): ")
    attendance_system.mark_attendance(student_id, date, status)
elif choice == '3':
    date = input("Enter date to display attendance for (YYYY-MM-DD): ")
    attendance_system.display_attendance_by_date(date)
elif choice == '4':
    student_id = input("Enter student ID to display attendance for: ")
    attendance_system.display_attendance_by_date(student_id)
elif choice == '5':
    attendance_system.display_full_attendance_report()
elif choice == '6':
    print("Exiting Attendance Management System. Goodbye!")
    break
else:
    print("invalid choice. Please enter a number between 1 and 6.")


attendance_system = AttendanceSystem()
attendance_system.main()

```

```

# Run the system
run_attendance_system()

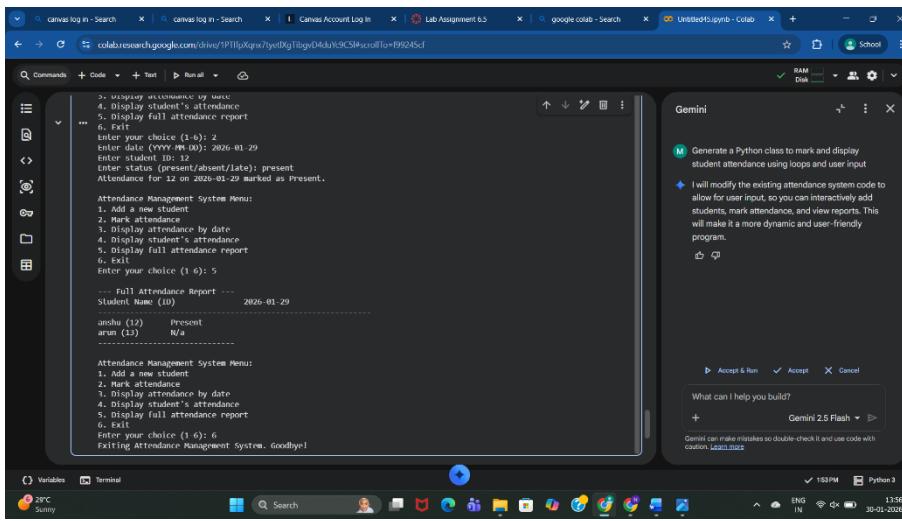
*** Attendance Management System Menu:
1. Add a new student
2. Mark attendance
3. Display attendance by date
4. Display student's attendance
5. Display full attendance report
6. Exit

Enter your choice (1-6): 1
Enter student ID: 13
Enter student name: arun
Student arun (ID: 13) added.

Attendance Management System Menu:
1. Add a new student
2. Mark attendance
3. Display attendance by date
4. Display student's attendance
5. Display full attendance report
6. Exit

attendance_system = AttendanceSystem()
attendance_system.main()

```



Correct Display of Attendance

- The system correctly displays attendance in three different ways:
 1. **By date** – Shows each student's attendance status for a given date.
 2. **By student** – Displays all attendance records (date and status) for a specific student.
 3. **Full attendance report** – Displays a tabular view of all students against all recorded dates.
 - Attendance status is validated and displayed in a **clear, readable format** (Present, Absent, Late).
 - If no records exist, the system gracefully handles cases by displaying appropriate messages (e.g., “*No attendance records found*”).

This confirms the attendance is stored, retrieved, and displayed correctly using loops and conditionals.

Test Cases

Test Case 1: Add Student

Input:

Student ID: 101

Name: Alice

Output:

Student Alice (ID: 101) added.

Test Case 2: Mark Attendance

Input:

Date: 2026-01-30

Student ID: 101

Status: present

Output:

Attendance for 101 on 2026-01-30 marked as Present.

Test Case 3: Display Attendance by Date

Input:

Date: 2026-01-30

Output:

--- Attendance for 2026-01-30 ---

Alice (ID: 101): Present

Test Case 4: Display Student Attendance

Input:

Student ID: 101

Output:

Date: 2026-01-30, Status: Present

Test Case 5: Full Attendance Report

Output:

Student Name (ID) 2026-01-30

Alice (101) Present

Observation:

In this task, AI-assisted code created a structured attendance management system using classes and loops. The program correctly added students, marked attendance, and displayed attendance records by date, by student, and as a complete report. Conditional statements ensured valid inputs and proper handling of missing records, resulting in accurate and well-organized attendance display.

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

Prompt

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

Code:

```
def atm_menu():

    balance = 1000 # Initial balance

    while True:

        print("\n--- ATM 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':
            try:
                deposit_amount = float(input("Enter amount to deposit: "))

                if deposit_amount > 0:
                    balance += deposit_amount
                    print(f"${deposit_amount:.2f} deposited successfully.")

                    print(f"New balance: ${balance:.2f}")

                else:
                    print("Deposit amount must be positive.")

            except ValueError:
                print("Invalid amount. Please enter a number.")

        elif choice == '3':
```

```
try:
```

```
    withdraw_amount = float(input("Enter amount to withdraw: "))
```

```
    if withdraw_amount > 0:
```

```
        if balance >= withdraw_amount:
```

```
            balance -= withdraw_amount
```

```
            print(f"${withdraw_amount:.2f} withdrawn successfully.")
```

```
            print(f"New balance: ${balance:.2f}")
```

```
        else:
```

```
            print("Insufficient funds.")
```

```
    else:
```

```
        print("Withdrawal amount must be positive.")
```

```
except ValueError:
```

```
    print("Invalid amount. 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.")
```

```
atm_menu()
```

The screenshot shows a Google Colab notebook with the following code in a cell:

```
def atm_menu():
    balance = 1000 # Initial balance

    while True:
        print("Welcome to ATM 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"You current balance is: ${balance:.2f}")
        elif choice == "2":
            try:
                deposit_amount = float(input("Enter amount to deposit: "))
                if deposit_amount > 0:
                    balance += deposit_amount
                    print(f"${deposit_amount:.2f} deposited successfully.")
                    print(f"New balance: ${balance:.2f}")
                else:
                    print("Deposit amount must be positive.")
            except ValueError:
                print("Invalid amount. Please enter a number.")
        elif choice == "3":
            try:
                withdraw_amount = float(input("Enter amount to withdraw: "))
                if withdraw_amount < balance:
                    balance -= withdraw_amount
                    print(f"${withdraw_amount:.2f} withdrawn successfully.")
                else:
                    print("Insufficient funds. Please enter a valid amount less than or equal to your current balance: ${balance:.2f}")
            except ValueError:
                print("Invalid amount. Please enter a number.")
        elif choice == "4":
            break
        else:
            print("Invalid choice. Please enter a number between 1 and 4.")

atm_menu()
```

To the right of the code cell, there is a Gemini AI interface window. It displays the following text:

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

Here's a Python program that simulates an ATM menu, allowing you to check balance, deposit, withdraw, and exit.

The Gemini interface includes buttons for "Accept & Run", "Accept", and "Cancel". A message at the bottom says: "Gemini can make mistakes so double-check it and use code with caution. Learn more".

The screenshot shows a Google Colab notebook titled "Untitled4.ipynb". The code in the cell is as follows:

```
print("Withdraw amount:.2f) withdrawn successfully.")
    print("New balance: ${balance:.2f}")
else:
    print("Insufficient funds.")

else:
    print("Withdrawal amount must be positive.")
except ValueError:
    print("Invalid amount. 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 menu simulation
atm_menu()
```

Output from the cell:

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

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

The sidebar on the right contains a Gemini AI interface with the following text:

M Generate a Python program using loops and conditionals to simulate an ATM menu

Here's a Python program that simulates an ATM menu, allowing you to check balance, deposit, withdraw, and exit.

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

The screenshot shows the same Google Colab notebook with the same code and output. The output is identical to the previous screenshot, demonstrating the correct handling of user input for the ATM menu options.

Correct Option Handling

- The program uses conditional statements (if–elif–else) to correctly handle all ATM menu options.
- **Option 1 (Check Balance):** Displays the current account balance.

- **Option 2 (Deposit Money):**
 - Accepts a valid positive amount.
 - Rejects negative or non-numeric inputs using try-except.
- **Option 3 (Withdraw Money):**
 - Allows withdrawal only if sufficient balance is available.
 - Prevents negative amounts and overdrafts.
- **Option 4 (Exit):** Safely terminates the program.
- Any invalid menu selection is handled gracefully with an error message.

This ensures reliable and user-safe navigation through the ATM menu.

Output Verification

Test Case 1: Check Balance

Input:

1

Output:

Your current balance is: \$1000.00

Test Case 2: Deposit Money

Input:

2

500

Output:

\$500.00 deposited successfully.

New balance: \$1500.00

Test Case 3: Withdraw Money (Valid)

Input:

3

300

Output:

\$300.00 withdrawn successfully.

New balance: \$1200.00

Test Case 4: Withdraw Money (Insufficient Funds)

Input:

3

2000

Output:

Insufficient funds.

Test Case 5: Invalid Option

Input:

7

Output:

Invalid choice. Please enter a number between 1 and 4.

Observation:

For the ATM menu simulation, AI-generated code correctly implemented a loop-based menu system with conditional option handling. The program accurately processed balance checks, deposits, withdrawals, and exit operations. Input validation and error handling ensured safe transactions and correct balance updates, demonstrating effective use of loops and conditionals.