

AI Assisted Coding

Name : M.Harish

H.no : 2303A52387

Batch : 34

Course Code : 23CS002PC304

Assignment Number : 6.5

Experiment 6: AI-Based Code Completion: Working with suggestions for classes, loops, conditionals

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

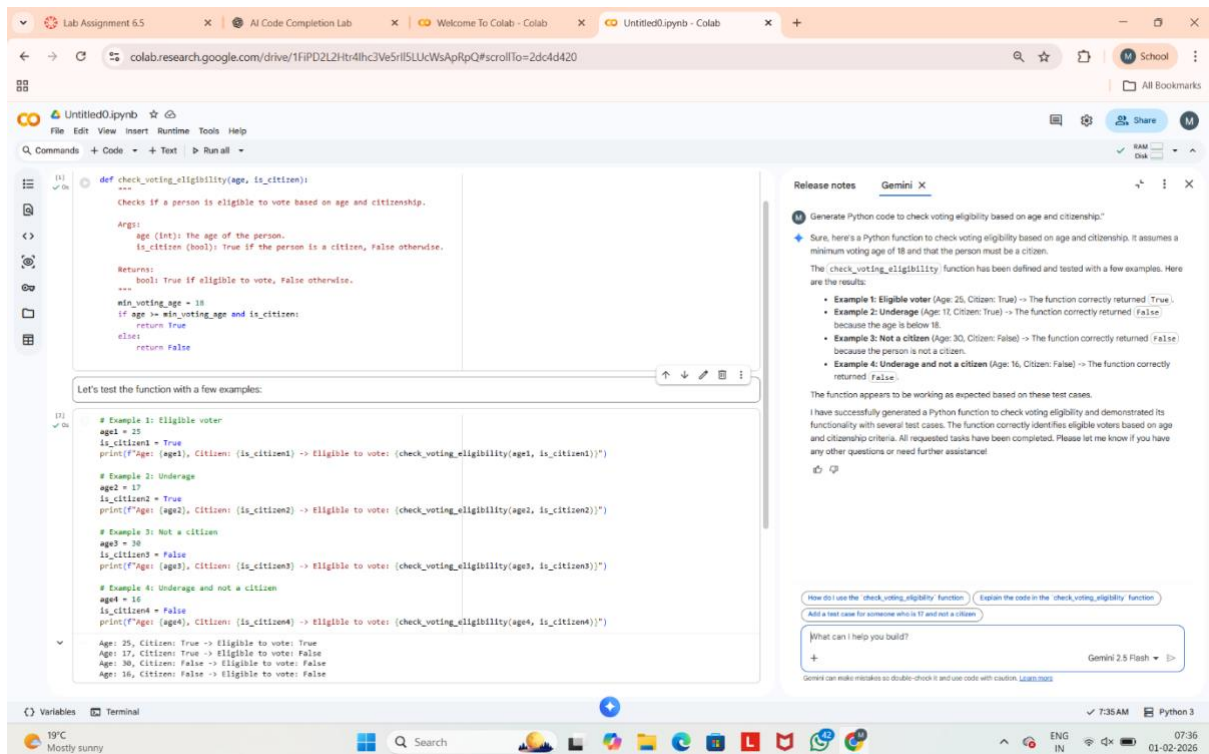
Task: Use an AI tool to generate eligibility logic.

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.



Explanation of Logic

- age takes integer input from the user
- citizen takes string input and converts it to lowercase
- The if condition checks:
 - Age must be **18 or above**
 - Citizenship must be **yes**
- If both conditions are true → eligible
- Otherwise → not eligible

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.

The screenshot shows a Google Colab notebook with the following content:

```
def count_vowels_consonants(input_string):  
    """  
    Counts the number of vowels and consonants in a string.  
    Args:  
        input_string (str): The string to analyze.  
    Returns:  
        tuple: A tuple containing (vowel_count, consonant_count).  
    """  
    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
```

Let's test the `count_vowels_consonants` function with an example:

```
test_string = "Hello World! Python is fun."  
v1, c1 = count_vowels_consonants(test_string)  
print(f"Original string: '{test_string}'")  
print(f"Number of vowels: {v1}")  
print(f"Number of consonants: {c1}")  
  
test_string_2 = "A quick brown fox jumps over the lazy dog."  
v2, c2 = count_vowels_consonants(test_string_2)  
print(f"Original string: '{test_string_2}'")  
print(f"Number of vowels: {v2}")  
print(f"Number of consonants: {c2}")
```

Output:

```
Original string: 'Hello World! Python is fun.'  
Number of vowels: 6  
Number of consonants: 15  
  
Original string: 'A quick brown fox jumps over the lazy dog.'  
Number of vowels: 11  
Number of consonants: 22
```

The Gemini AI assistant provides the following explanation:

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

Certainly! Here's a Python function that counts the number of vowels and consonants in a given string using a loop. It considers only alphabetic characters and treats 'y' as a consonant for this purpose.

The `count_vowels_consonants` function has been successfully executed with two example strings, and here are the results:

- For the string `"Hello World! Python is fun."`:
 - Number of vowels: 6
 - Number of consonants: 15
- For the string `"A quick brown fox jumps over the lazy dog."`:
 - Number of vowels: 11
 - Number of consonants: 22

The function appears to be working correctly, accurately counting vowels and consonants in the provided strings.

I have successfully generated and tested Python code for both checking voting eligibility and counting vowels and consonants in a string. All requested tasks have been completed. Please let me know if you have any other questions or need further assistance!

Explanation

- Converts string to lowercase for uniform comparison
- Uses a for loop to traverse each character
- `isalpha()` ensures only letters are counted
- Characters are classified as **vowels** or **consonants**

Task Description #3 (AI-Assisted Code Completion Reflection

Task)

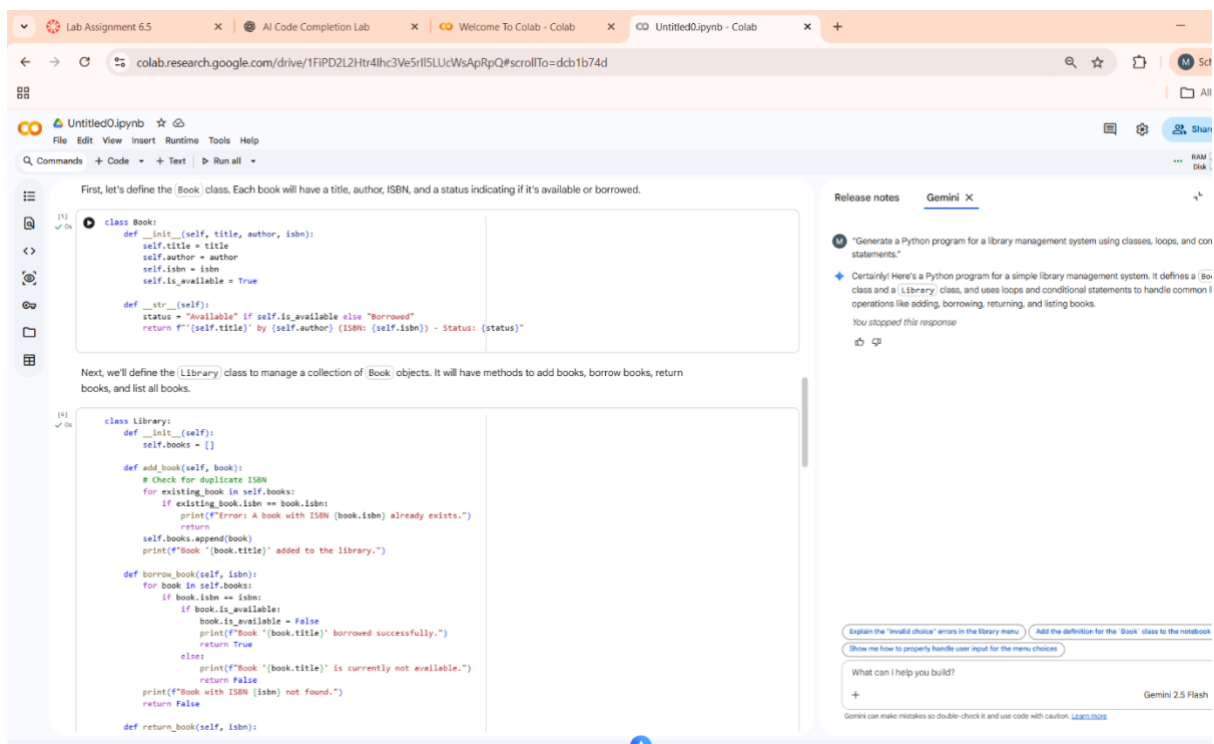
Task: Use an AI tool to generate a complete program using classes, loops, and conditionals.

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.



Lab Assignment 6.5 | AI Code Completion Lab | Welcome To Colab - Colab | Untitled0.ipynb - Colab

colab.research.google.com/drive/1FIPD2L2Htr4Ihc3Ve5nI5LUcWsApRpQ#scrollTo=dc61b74d

Untitled0.ipynb

File Edit View Insert Runtime Tools Help

Commands | Code | Text | Run all

First, let's define the `Book` class. Each book will have a title, author, ISBN, and a status indicating if it's available or borrowed.

```
[1] class Book:
    def __init__(self, title, author, isbn):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.is_available = True

    def __str__(self):
        status = "available" if self.is_available else "borrowed"
        return f"{self.title} by {self.author} (ISBN: {self.isbn}) - Status: {status}"
```

Next, we'll define the `Library` class to manage a collection of `Book` objects. It will have methods to add books, borrow books, return books, and list all books.

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

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

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

    def return_book(self, isbn):
```

Release notes | Gemini X

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

✦ Certainly! Here's a Python program for a simple library management system. It defines a `Book` class and a `Library` class, and uses loops and conditional statements to handle common library operations like adding, borrowing, returning, and listing books.

You stopped this response

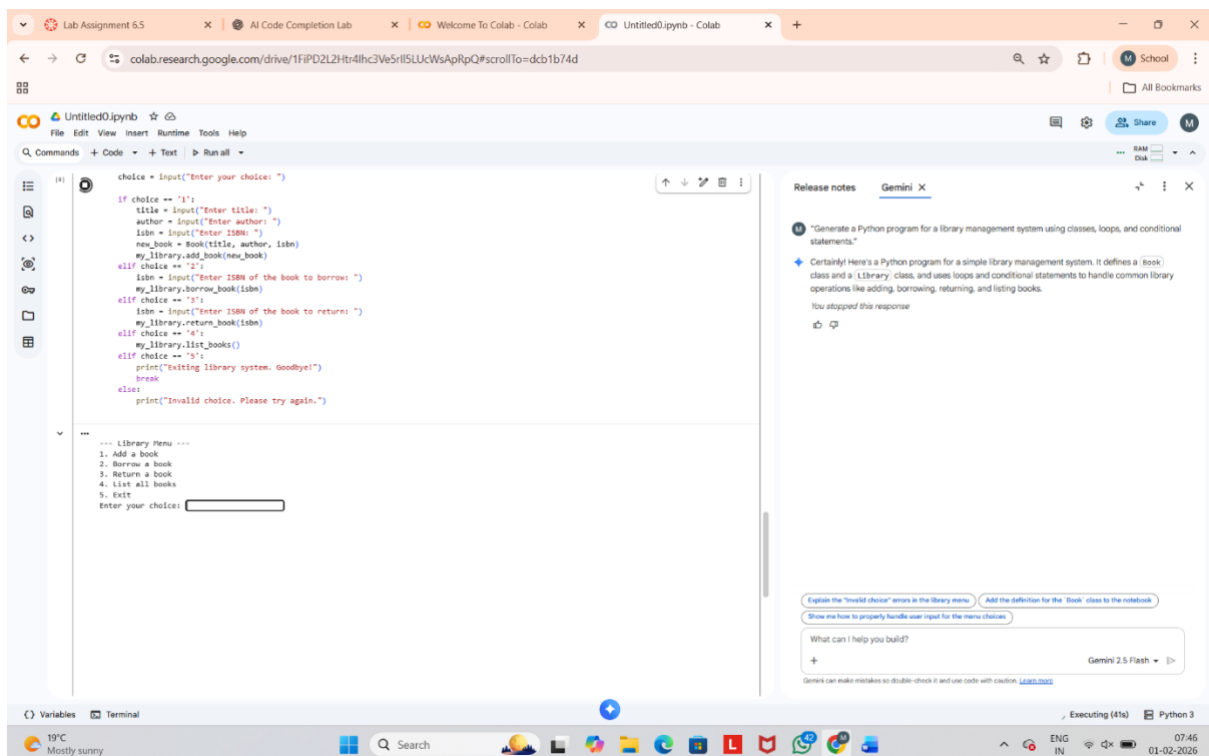
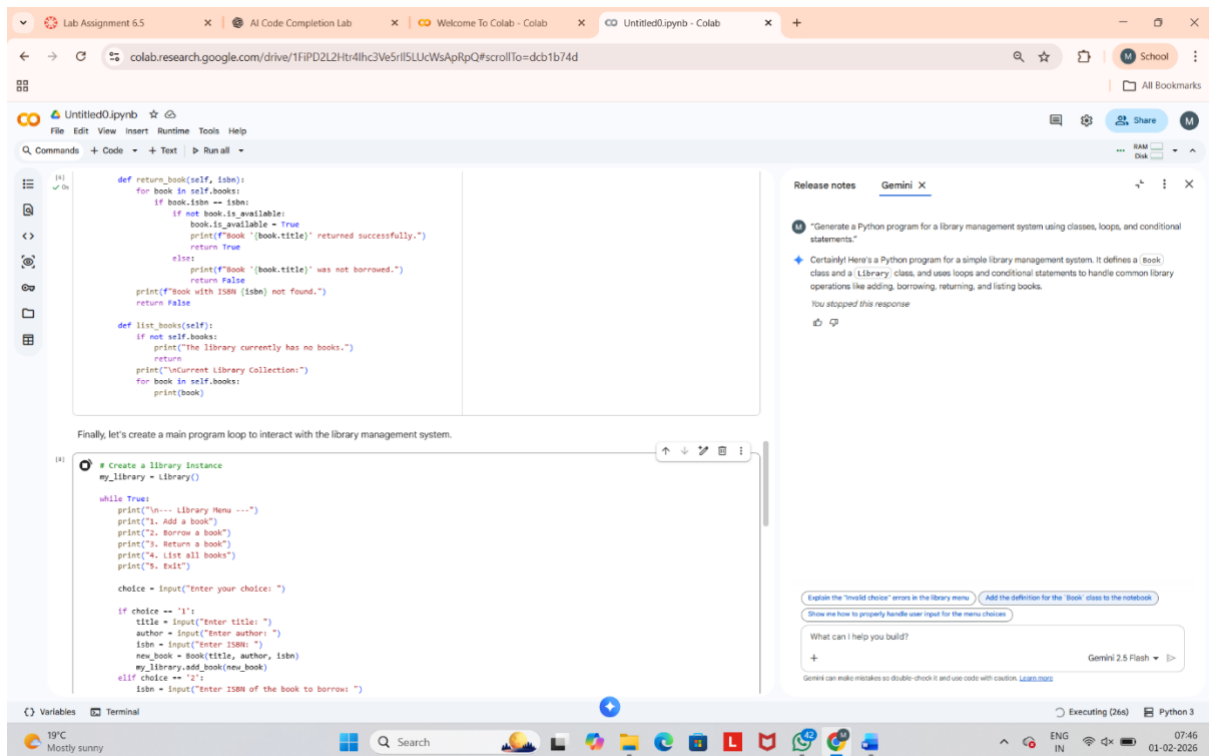
Explain the "Invalid choice" errors in the library menu | Add the definition for the 'Book' class to the notebook | Show me how to properly handle user input for the menu choices

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](#)



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.

The screenshot displays a Google Colab notebook titled 'Untitled0.ipynb'. The main code cell contains a Python class named `AttendanceSystem` designed to manage student names and their attendance records. The class includes methods for initialization, marking attendance, displaying attendance for a specific date, and displaying a full attendance report. Comments within the code explain the logic, such as ensuring unique and sorted student names and handling default attendance status.

```
Let's define the AttendanceSystem class. It will manage student names and their attendance records.

class AttendanceSystem:
    def __init__(self, student_names):
        self.student_names = sorted(list(set(student_names))) # Ensure unique and sorted student names
        self.attendance_records = {}
        print(f"Attendance system initialized for students: {', '.join(self.student_names)}")

    def mark_attendance(self, date, attendance_data): # attendance_data should be a dict like {'student_name': 'P' or 'A'}
        if date in self.attendance_records:
            print(f"Warning: Attendance for {date} already exists. Overwriting.")

        current_day_attendance = {}
        for student in self.student_names:
            status = attendance_data.get(student, 'A') # Default to Absent if not provided
            if status not in ['P', 'A']:
                print(f"Invalid attendance status '{status}' for {student} on {date}. Setting to 'A'.")
            status = 'A'
            current_day_attendance[student] = status

        self.attendance_records[date] = current_day_attendance
        print(f"Attendance for {date} marked successfully.")

    def display_attendance(self, student_name=None):
        if not self.attendance_records:
            print("No attendance records available.")
            return

        if student_name and student_name not in self.student_names:
            print(f"Error: Student '{student_name}' not found in the system.")
            return

        dates = sorted(self.attendance_records.keys())

        if student_name:
            print(f"\n--- Attendance for {student_name} ---")
            for date in dates:
                status = self.attendance_records[date].get(student_name, 'N/A') # N/A if student not in record for that day
                print(f"    {date}: {status}")
        else:
            print("\n--- Full Attendance Report ---")
            # Print header row
            header = "Date\t" + "\t".join(self.student_names)
            print(header)
            for date in dates:
                row = date + "\t"
                for student in self.student_names:
                    row += self.attendance_records[date].get(student, 'N/A') + "\t"
                print(row)
            print("\n---")
```

On the right side of the notebook, the 'Release notes' for Gemini are displayed. They include a summary of the AI's capabilities and a list of features demonstrated in the code, such as initialization, marking attendance, and generating reports. The AI also provides a brief explanation of how the code integrates the `AttendanceSystem` class.

Lab Assignment 6.5 | AI Code Completion Lab | Welcome To Colab - Colab | Untitled0.ipynb - Colab

colab.research.google.com/drive/1FIPD2L2Htr4lhC3Ve5rII5LUcWsApRpQ?scrollTo=dc1b74d

Untitled0.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

```
[10] # Print attendance for each date
for date in dates:
    row = f'{date}\n'
    for student in self.student_names:
        status = self.attendance_records[date].get(student, 'N/A')
        row += f'{status}\n'
    print(row.strip())
```

Let's create an instance of the `AttendanceSystem` and demonstrate its functionality.

```
[11] # Initialize the system with student names
students = ["Alice", "Bob", "Charlie", "David"]
attendance_tracker = AttendanceSystem(students)

# Mark attendance for a few dates
print("Marking attendance for 2023-10-26:")
attendance_tracker.mark_attendance("2023-10-26", {
    "Alice": "P",
    "Bob": "P",
    "Charlie": "A",
    "David": "P"
})

print("Marking attendance for 2023-10-27:")
attendance_tracker.mark_attendance("2023-10-27", {
    "Alice": "A",
    "Bob": "P",
    "Charlie": "P",
    "David": "A"
})

print("Marking attendance for 2023-10-28:")
attendance_tracker.mark_attendance("2023-10-28", {
    "Alice": "P",
    "Bob": "A",
    "Charlie": "P",
    "David": "P"
})

# Display full attendance report
attendance_tracker.display_attendance()

# Display attendance for a specific student
attendance_tracker.display_attendance("Alice")
```

Release notes Gemini

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

Certainly! Here's a Python class for a student attendance system. You can initialize it with a list of student names, mark attendance for specific dates, and then view the attendance records. The `AttendanceSystem` class has been successfully defined and demonstrated. Here's a summary of its functionality:

- Initialization:** An attendance system was initialized for students: Alice, Bob, Charlie, David.
- Marking Attendance:** Attendance was marked for three dates (2023-10-26, 2023-10-27, 2023-10-28) with different present (P) and absent (A) statuses for each student.
- Full Attendance Report:** A full attendance report was displayed, showing the status for all students across all marked dates.
- Individual Student Attendance:** Attendance records were displayed for 'Alice' and 'Bob' separately.
- Overwriting Attendance:** The attendance for 2023-10-26 was marked again with changes (Bob now absent, Charlie now present). The system correctly issued a warning about overwriting and then updated the record.
- Updated Full Attendance Report:** A final full report showed the updated attendance for 2023-10-26.

The system is working as expected, demonstrating the ability to track and display student attendance, including handling updates to existing records.

I have successfully generated and tested Python code for checking voting eligibility, counting vowels and consonants in a string, creating a library management system, and implementing a student attendance system. All your requests have been completed. Please let me know if you have any other questions or need further assistance!

Initialize and use the `AttendanceSystem` class Add a method to the `Library` class to find a book by title Explain how to integrate the `Library` and `AttendanceSystem` classes

What can I help you build?

+ Gemini 2.5 Flash >

7:50 AM Python 3

Lab Assignment 6.5 | AI Code Completion Lab | Welcome To Colab - Colab | Untitled0.ipynb - Colab

colab.research.google.com/drive/1FIPD2L2Htr4lhC3Ve5rII5LUcWsApRpQ?scrollTo=dc1b74d

Untitled0.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text + Run all

```
[11] # Try to mark attendance for an existing date (should overwrite)
print("Marking attendance for 2023-10-26 again with changes!")
attendance_tracker.mark_attendance("2023-10-26", {
    "Alice": "P",
    "Bob": "A", # Bob is now absent
    "Charlie": "P", # Charlie is now present
    "David": "P"
})

# Display full attendance report again to see the changes
attendance_tracker.display_attendance()
```

Attendance system initialized for students: Alice, Bob, Charlie, David

Marking attendance for 2023-10-26:
Attendance for 2023-10-26 marked successfully.

Marking attendance for 2023-10-27:
Attendance for 2023-10-27 marked successfully.

Marking attendance for 2023-10-28:
Attendance for 2023-10-28 marked successfully.

--- Full Attendance Report ---

Date	Alice	Bob	Charlie	David
2023-10-26	P	P	A	P
2023-10-27	A	P	P	A
2023-10-28	P	A	P	P

--- Attendance for Alice ---

Date	Status
2023-10-26	P
2023-10-27	A
2023-10-28	P

--- Attendance for Bob ---

Date	Status
2023-10-26	P
2023-10-27	P
2023-10-28	A

Marking attendance for 2023-10-26 again with changes:
Warning: Attendance for 2023-10-26 already exists. Overwriting.
Attendance for 2023-10-26 marked successfully.

--- Full Attendance Report ---

Date	Alice	Bob	Charlie	David
2023-10-26	P	A	P	P
2023-10-27	A	P	P	A
2023-10-28	P	A	P	P

Release notes Gemini

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

Certainly! Here's a Python class for a student attendance system. You can initialize it with a list of student names, mark attendance for specific dates, and then view the attendance records. The `AttendanceSystem` class has been successfully defined and demonstrated. Here's a summary of its functionality:

- Initialization:** An attendance system was initialized for students: Alice, Bob, Charlie, David.
- Marking Attendance:** Attendance was marked for three dates (2023-10-26, 2023-10-27, 2023-10-28) with different present (P) and absent (A) statuses for each student.
- Full Attendance Report:** A full attendance report was displayed, showing the status for all students across all marked dates.
- Individual Student Attendance:** Attendance records were displayed for 'Alice' and 'Bob' separately.
- Overwriting Attendance:** The attendance for 2023-10-26 was marked again with changes (Bob now absent, Charlie now present). The system correctly issued a warning about overwriting and then updated the record.
- Updated Full Attendance Report:** A final full report showed the updated attendance for 2023-10-26.

The system is working as expected, demonstrating the ability to track and display student attendance, including handling updates to existing records.

I have successfully generated and tested Python code for checking voting eligibility, counting vowels and consonants in a string, creating a library management system, and implementing a student attendance system. All your requests have been completed. Please let me know if you have any other questions or need further assistance!

Initialize and use the `AttendanceSystem` class Add a method to the `Library` class to find a book by title Explain how to integrate the `Library` and `AttendanceSystem` classes

What can I help you build?

+ Gemini 2.5 Flash >

7:50 AM Python 3

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.

The screenshot shows a Google Colab notebook titled 'Untitled0.ipynb'. The code in the notebook is a Python program for an ATM simulation. It starts with an initial balance of 1000. The program uses a while loop to keep the menu running. The menu options are: 1. Check Balance, 2. Deposit, 3. Withdraw, and 4. Exit. The program handles each option with conditional statements and loops. For example, for the deposit option, it asks for the deposit amount and checks if it's positive. For the withdraw option, it asks for the withdraw amount and checks if it's positive and if there are sufficient funds. The program ends with a 'Thank you for using the ATM. Goodbye!' message when the user selects the Exit option.

```
Let's create a Python program to simulate an ATM menu. This program will allow users to check their balance, deposit money, withdraw money, and exit.

account_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: ${account_balance:.2f}")
    elif choice == '2':
        try:
            deposit_amount = float(input("Enter amount to deposit: "))
            if deposit_amount > 0:
                account_balance += deposit_amount
                print(f"${deposit_amount:.2f} deposited successfully.")
                print(f"New balance: ${account_balance:.2f}")
            else:
                print("Deposit amount must be positive.")
        except ValueError:
            print("Invalid input. Please enter a number.")
    elif choice == '3':
        try:
            withdraw_amount = float(input("Enter amount to withdraw: "))
            if withdraw_amount <= 0:
                print("Withdrawal amount must be positive.")
            elif withdraw_amount > account_balance:
                print("Insufficient funds.")
            else:
                account_balance -= withdraw_amount
                print(f"${withdraw_amount:.2f} withdrawn successfully.")
                print(f"New balance: ${account_balance:.2f}")
        except ValueError:
            print("Invalid input. Please enter a number.")
    elif choice == '4':
        print("Thank you for using the ATM. Goodbye!")
        break # Exit the loop
```

Release notes Gemini X

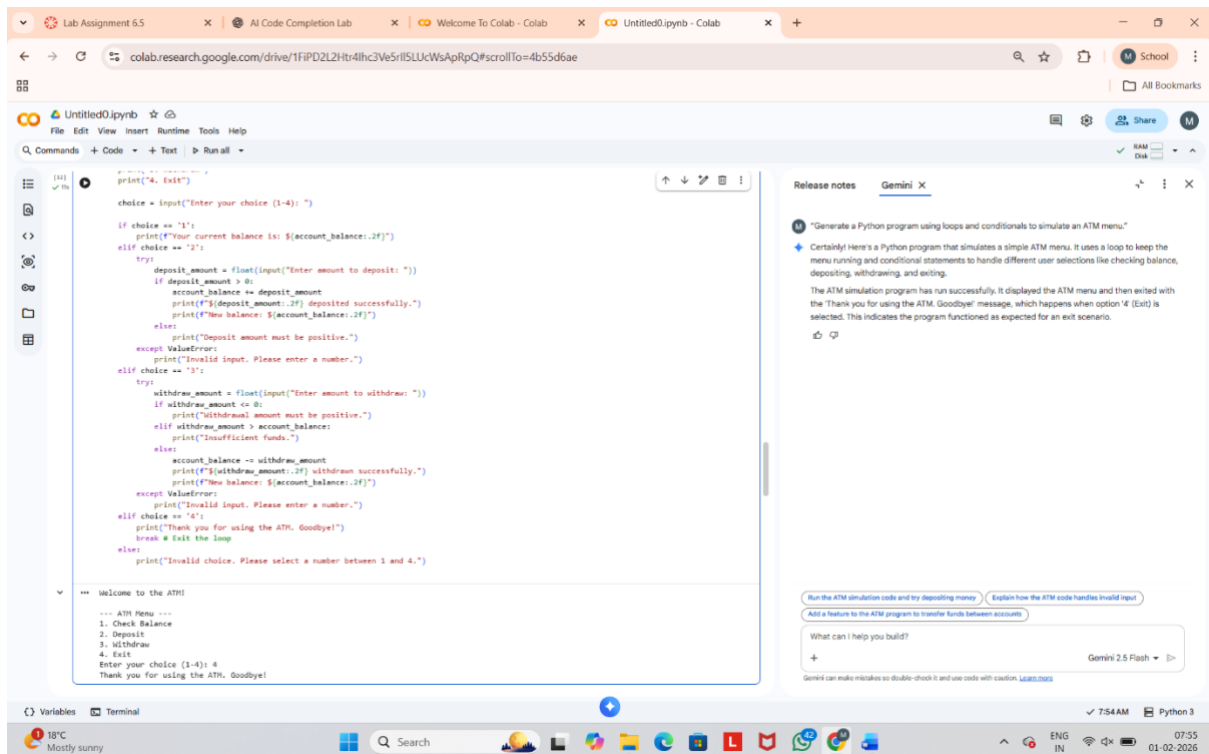
"Generate a Python program using loops and conditionals to simulate an ATM menu."

Certainly! Here's a Python program that simulates a simple ATM menu. It uses a loop to keep the menu running and conditional statements to handle different user selections like checking balance, depositing, withdrawing, and exiting.

The ATM simulation program has run successfully. It displayed the ATM menu and then exited with the 'Thank you for using the ATM. Goodbye!' message, which happens when option '4' (Exit) is selected. This indicates the program functioned as expected for an exit scenario.

What can I help you build?

Gemini 2.5 Flash



Conclusion

This experiment demonstrates effective use of AI-based code completion for generating Python programs involving **classes, loops, and conditionals**. AI tools enhance productivity, but human judgment is essential for correctness, optimization, and ethical usage.