

# **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.

```

def check_voting_eligibility(age, is_citizen):
    """
    Checks if a person is eligible to vote based on age and citizenship.

    Args:
        age (int): The age of the person.
        is_citizen (bool): True if the person is a citizen, False otherwise.

    Returns:
        bool: True if eligible to vote, False otherwise.
    """
    min_voting_age = 18
    if age >= min_voting_age and is_citizen:
        return True
    else:
        return False

```

Let's test the function with a few examples:

```

# Example 1: Eligible voter
age1 = 25
is_citizen1 = True
print(f"Age: {age1}, Citizen: {is_citizen1} -> Eligible to vote: {check_voting_eligibility(age1, is_citizen1)}")

# Example 2: Underage
age2 = 12
is_citizen2 = True
print(f"Age: {age2}, Citizen: {is_citizen2} -> Eligible to vote: {check_voting_eligibility(age2, is_citizen2)}")

# Example 3: Not a citizen
age3 = 30
is_citizen3 = False
print(f"Age: {age3}, Citizen: {is_citizen3} -> Eligible to vote: {check_voting_eligibility(age3, is_citizen3)}")

# Example 4: Underage and not a citizen
age4 = 16
is_citizen4 = False
print(f"Age: {age4}, Citizen: {is_citizen4} -> Eligible to vote: {check_voting_eligibility(age4, is_citizen4)}")

```

Age: 25, Citizen: True -> Eligible to vote: True  
Age: 12, Citizen: True -> Eligible to vote: False  
Age: 30, Citizen: False -> Eligible to vote: False  
Age: 16, Citizen: False -> Eligible to vote: False

**Release notes** Gemini X

Generate Python code to check voting eligibility based on age and citizenship.  
Sure, here's a Python function to check voting eligibility based on age and citizenship. It assumes a minimum voting age of 18 and that the person must be a citizen.  
The `check_voting_eligibility` function has been defined and tested with a few examples. Here are the results:  
• Example 1: Eligible voter (Age: 25, Citizen: True) -> The function correctly returned `True`.  
• Example 2: Underage (Age: 12, Citizen: True) -> The function correctly returned `False` because the age is below 18.  
• Example 3: Not a citizen (Age: 30, Citizen: False) -> The function correctly returned `False` because the person is not a citizen.  
• Example 4: Underage and not a citizen (Age: 16, Citizen: False) -> The function correctly returned `False`.  
The function appears to be working as expected based on these test cases.  
I have successfully generated a Python function to check voting eligibility and demonstrated its functionality with several test cases. The function correctly identifies eligible voters based on age and citizenship criteria. All requested tasks have been completed. Please let me know if you have any other questions or need further assistance!

How do I use the `check_voting_eligibility` function? Explain the code in the `check_voting_eligibility` function.  
Add a test case for someone who is 17 and not a citizen.

What can I help you build?

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

Variables Terminal 19°C Mostly sunny 7:35 AM Python 3 ENG IN 01-02-2026

## 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 interface with several tabs at the top: "Lab Assignment 6.5", "AI Code Completion Lab", "Welcome To Colab - Colab", and "Untitled0.ipynb - Colab". The main area displays a Python script named "Untitled0.ipynb".

```
def count_vowels_and_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
```

Below the code, a comment says: "Let's test the `count_vowels_and_consonants` function with an example:"

```
[4]: test_string = "Hello World! Python is fun."
v, c = count_vowels_and_consonants(test_string)
print("Original string: ", test_string)
print("Number of vowels: ", v)
print("Number of consonants: ", c)

test_string_2 = "A quick brown fox jumps over the lazy dog."
v2, c2 = count_vowels_and_consonants(test_string_2)
print("Original string: ", test_string_2)
print("Number of vowels: ", v2)
print("Number of consonants: ", c2)
```

The output shows the results for both strings:

```
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
```

To the right of the code editor, there is a sidebar titled "Gemini" which provides release notes and a summary of the generated code's functionality. It also includes a "Generate code" button and a "Gemini 2.5 Flash" button.

## 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.

The screenshot shows a Google Colab notebook titled "Untitled0.ipynb". The code defines a `Book` class with an `__init__` method and a `__str__` method. It also defines a `Library` class with methods for adding books, borrowing books, and returning books. The AI has provided release notes and a Gemini interface for further interaction.

```
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}"

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("Error: A book with ISBN {} already exists.".format(book.isbn))
                return
        self.books.append(book)
        print("Book '{}' added to the library.".format(book.title))

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

    def return_book(self, isbn):
        pass
```

Release notes

Gemini

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

Gemini! 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 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](#)

```

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

    def borrow(self):
        if not self.is_borrowed:
            print(f"Book '{self.title}' borrowed successfully.")
            self.is_borrowed = True
        else:
            print("Book '{self.title}' was not borrowed.")

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

    def list_books(self):
        if not self.books:
            print("The library currently has no books.")
            return
        print("\nCurrent Library Collection:")
        for book in self.books:
            print(book)

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

    def add_book(self, book):
        self.books.append(book)

    def borrow_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                book.borrow()
                break
            else:
                print("Book with ISBN {isbn} not found.")

    def return_book(self, isbn):
        for book in self.books:
            if book.isbn == isbn:
                book.return_book(isbn)
                break
            else:
                print("Book with ISBN {isbn} not found.")

    def list_all_books(self):
        self.list_books()

# Create a library instance
my_library = Library()

while True:
    print("\n--- Library Menu ---")
    print("1. Add a book")
    print("2. Borrow a book")
    print("3. Return a book")
    print("4. List all books")
    print("5. 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':
        isbn = input("Enter ISBN of the book to borrow: ")
        my_library.borrow_book(isbn)
    elif choice == '3':
        isbn = input("Enter ISBN of the book to return: ")
        my_library.return_book(isbn)
    elif choice == '4':
        my_library.list_all_books()
    elif choice == '5':
        print("Exiting library system. Goodbye!")
        break
    else:
        print("Invalid choice. Please try again.")


```

```

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':
    isbn = input("Enter ISBN of the book to borrow: ")
    my_library.borrow_book(isbn)
elif choice == '3':
    isbn = input("Enter ISBN of the book to return: ")
    my_library.return_book(isbn)
elif choice == '4':
    my_library.list_all_books()
elif choice == '5':
    print("Exiting library system. Goodbye!")
    break
else:
    print("Invalid choice. Please try again.")

... Library Menu ...
1. Add a book
2. Borrow a book
3. Return a book
4. List all books
5. Exit
Enter your choice: [REDACTED]


```

## 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 shows a Google Colab interface with two tabs open: "Lab Assignment 6.5" and "Untitled0.ipynb". The "Untitled0.ipynb" tab is active, displaying the following Python code:

```
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("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("Warning: Attendance for (date) already exists. Overwriting.")

        current_day_attendance = {}
        for student_name 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("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".join(self.student_names)
            print(header)
            for date in dates:
                row = [date]
                for student_name in self.student_names:
                    status = self.attendance_records.get(date, {}).get(student_name, 'N/A')
                    row.append(status)
                print("\t".join(row))

    def __str__(self):
        return f"Attendance System for {len(self.student_names)} students: {', '.join(self.student_names)}"
```

The code defines a class `AttendanceSystem` that initializes with a list of student names and can mark attendance for specific dates. It also provides a method to display attendance for individual students or for all students across all dates. The AI has added comments and error handling to the original code.

The screenshot shows a Google Colab interface with the following details:

- Title Bar:** Lab Assignment 6.5, AI Code Completion Lab, Welcome To Colab - Colab, Untitled0.ipynb - Colab.
- Code Editor:**

```
# Print attendance for each date
for date in dates:
    row = f"({date})"
    for student_name in student_names:
        status = self.attendance_records[date].get(student, 'N/A')
        row += f"\t{status}"
    print(row.strip())

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

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

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

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

print("\nMarking 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:**

"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.
- Bottom Status:** Gemini 2.5 Flash, 7:50AM, Python 3, ENG IN, 01-02-2026.

The screenshot shows a Google Colab interface with the following details:

- Title Bar:** Lab Assignment 6.5, AI Code Completion Lab, Welcome To Colab - Colab, Untitled0.ipynb - Colab.
- Code Editor:**

```
# Try to mark attendance for an existing date (should overwrite)
print("\nMarking 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
Attendance for 2023-10-26:
Attendance for 2023-10-26 marked successfully.

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

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 ---
2023-10-26: P
2023-10-27: A
2023-10-28: P

--- Attendance for Bob ---
2023-10-26: P
2023-10-27: P
2023-10-28: A

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 A P P P
2023-10-27 A P P A
2023-10-28 P A P P
```
- Release Notes:**

"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.
- Bottom Status:** Gemini 2.5 Flash, 7:50AM, Python 3, ENG IN, 01-02-2026.

## 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 is as follows:

```
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
```

The AI notes section contains the following text:

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

At the bottom right, there is a "Gemini 2.5 Flash" button and a note: "Gemini can make mistakes so double-check it and use code with caution." The status bar at the bottom shows the date and time as 01-02-2026, 7:54 AM, and Python 3.

```

print("4. Exit")

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

if choice == "1":
    print("Your current balance is: ${account_balance:.2f}")
elif choice == "2":
    try:
        deposit_amount = float(input("Enter amount to deposit: "))
        if deposit_amount > 0:
            account_balance += deposit_amount
            print("${deposit_amount:.2f} deposited successfully.")
            print("New balance: ${account_balance:.2f}")
        else:
            print("Deposit amount must be positive.")
    except ValueError:
        print("Invalid input. Please enter a number.")
elif choice == "3":
    try:
        withdraw_amount = float(input("Enter amount to withdraw: "))
        if withdraw_amount <= 0:
            print("Withdrawal amount must be positive.")
        elif withdraw_amount > account_balance:
            print("Insufficient funds.")
        else:
            account_balance -= withdraw_amount
            print("${withdraw_amount:.2f} withdrawn successfully.")
            print("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
else:
    print("Invalid choice. Please select a number between 1 and 4.")

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

```

Release notes Gemini X

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

Blue arrow icon: 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.

Run the ATM simulation code and try depositing money Explain how the ATM code handles invalid input

Add a feature to the ATM program to transfer funds between accounts

What can I help you build?

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

7:54 AM 7:55 ENG IN 01-02-2026

## 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.