

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

Hall Ticket No:2303A51846

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

Task: Use an AI tool to generate eligibility logic.

**Prompt:**

```
#Generate a python code that to check voting eligibility based on age and citizenship status.  
#include necessary comments to explain the code and correct eligibility criteria
```

**Code:**

```
def check_voting_eligibility(age, is_citizen):  
    """  
        This function checks if a person is eligible to vote based on their age and citizenship status.  
  
        Parameters:  
        age (int): The age of the person.  
        is_citizen (bool): True if the person is a citizen, False otherwise.  
  
        Returns:  
        bool: True if the person is eligible to vote, False otherwise.  
    """  
  
        # Voting eligibility criteria:  
        # 1. The person must be at least 18 years old.  
        # 2. The person must be a citizen.  
  
        if age >= 18 and is_citizen:  
            return True  
        else:  
            return False  
    # Example usage:  
    age = int(input("Enter your age: "))  
    is_citizen_input = input("Are you a citizen? (yes/no): ").strip().lower()  
    is_citizen = True if is_citizen_input == 'yes' else False  
    if check_voting_eligibility(age, is_citizen):  
        print("You are eligible to vote.")  
    else:  
        print("You are not eligible to vote.")
```

**Output:**

```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistant coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistant coding\Assignment(6.5)\voting.py"  
Enter your age: 21  
Are you a citizen? (yes/no): yes  
You are eligible to vote.  
PS C:\Users\madhu\OneDrive\Desktop\AI Assistant coding> █
```

**Observation:**

1. The code correctly applies voting eligibility rules by checking if the age is 18 or above and verifying citizenship status. This ensures the decision follows standard legal voting requirements.
2. A separate function is used to check eligibility, which makes the code modular and reusable. This improves readability and allows easy testing or future updates.
3. The function includes a detailed docstring explaining its purpose, inputs, and output. This helps other developers understand and maintain the code easily.
4. User input for citizenship is processed using strip() and lower() to avoid case-related errors. This makes the program more robust and user-friendly.
5. Boolean logic is used effectively to evaluate eligibility in a single conditional statement. This keeps the logic simple, clear, and efficient.
6. The program provides clear messages indicating whether the user is eligible to vote or not. This improves the overall user experience and clarity of results.

### **Task Description #2(AI-Based Code Completion for Loop-Based String Processing)**

**Task: Use an AI tool to process strings using loops.**

**Prompt:**

#Generate a python code to count vowels and consonants in a given string using loops.

**Code:**

```
def count_vowels_and_consonants(input_string):
    """
    This function counts the number of vowels and consonants in a given string.

    Parameters:
    input_string (str): The string to be analyzed.

    Returns:
    tuple: A tuple containing the count of vowels and consonants.
    """

    # Define vowels
    vowels = "aeiouAEIOU"

    # Initialize counts
    vowel_count = 0
    consonant_count = 0

    # Loop through each character in the string
    for char in input_string:
        # Check if the character is an alphabet
        if char.isalpha():
            # Check if the character is a vowel
            if char in vowels:
                |   vowel_count += 1
            else:
                |   consonant_count += 1

    return vowel_count, consonant_count
# Example usage:
input_string = input("Enter a string: ")
vowels, consonants = count_vowels_and_consonants(input_string)
print(f"Number of vowels: {vowels}")
print(f"Number of consonants: {consonants}")
```

**Output:**

```
PS C:\Users\madhu\OneDrive\Desktop\AI Assisted coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assisted coding\Assignment(6.5)\vowels.py"
Enter a string: AI Assisted coding
Number of vowels: 7
Number of consonants: 9
PS C:\Users\madhu\OneDrive\Desktop\AI Assisted coding>
```

### Observation:

1. The program correctly identifies and counts vowels and consonants by iterating through each character using a loop. This ensures every character in the string is analyzed properly.
2. It checks only alphabetic characters using `isalpha()`, ignoring numbers and special symbols. This prevents incorrect counting and improves accuracy.
3. Vowels are clearly defined in a separate string containing both uppercase and lowercase letters. This allows the program to handle case-insensitive vowel detection.
4. The use of separate counters for vowels and consonants makes the logic easy to understand. This also helps in maintaining and debugging the code.
5. The function is modular and reusable, returning results as a tuple. This allows the function to be used in other programs without modification.
6. Clear input and output statements make the program user-friendly. The final results are displayed in an understandable format for the user.

### 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 code to manage a library system using classes,loops and conditionalal statements.

#### Code:

```
class Book:
    """
    This class represents a book in the library.

    Attributes:
        title (str): The title of the book.
        author (str): The author of the book.
        available (bool): Availability status of the book.
    """

    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.available = True

    def borrow(self):
        """Marks the book as borrowed if it is available."""
        if self.available:
            self.available = False
            return True
        else:
            return False

    def return_book(self):
        """Marks the book as returned."""
        self.available = True
class Library:
    """
    This class represents a library system.

    Attributes:
        books (list): A list of Book objects in the library.
    """

```

```

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

def add_book(self, book):
    """Adds a book to the library."""
    self.books.append(book)

def display_books(self):
    """Displays all books in the library with their availability status."""
    for book in self.books:
        status = "Available" if book.available else "Borrowed"
        print(f"Title: {book.title}, Author: {book.author}, Status: {status}")

def borrow_book(self, title):
    """Allows a user to borrow a book by title."""
    for book in self.books:
        if book.title == title:
            if book.borrow():
                print(f"You have successfully borrowed '{title}'")
            else:
                print(f"Sorry, '{title}' is currently not available.")
    return

print(f"Book '{title}' not found in the library.")

def return_book(self, title):
    """Allows a user to return a book by title."""
    for book in self.books:
        if book.title == title:
            book.return_book()
            print(f"You have successfully returned '{title}'")
    return

print(f"Book '{title}' not found in the library.")

# Example usage:

```

```

# Example usage:
library = Library()
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("The Great Gatsby", "F. Scott Fitzgerald"))
library.display_books()
library.borrow_book("1984")
library.display_books()
library.return_book("1984")
library.display_books()
library.borrow_book("The Great Gatsby")
library.display_books()
library.return_book("The Great Gatsby")
library.display_books()
library.borrow_book("Moby Dick")
library.display_books()
library.return_book("Moby Dick")
library.display_books()

```

## **Output:**

```
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistent coding\Assignment(6.5)\library.py"
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have successfully borrowed '1984'.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Borrowed
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have successfully returned '1984'.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have successfully borrowed 'The Great Gatsby'.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have successfully returned 'The Great Gatsby'.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
Book 'Moby Dick' not found in the library.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
Book 'Moby Dick' not found in the library.
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
PS C:\Users\madhu\OneDrive\Desktop\AI Assistent coding>
```

## **Observation:**

1. The program uses Object-Oriented Programming with Book and Library classes. This makes the code structured and easy to maintain.
2. Book availability is managed using a boolean variable. It ensures a book cannot be borrowed twice at the same time.
3. Borrow and return operations are implemented using separate methods. This improves clarity and reusability of the code.
4. The library searches books using their title. It allows users to easily borrow or return a specific book.
5. Proper messages are displayed for success and failure cases. This helps users understand the system's response clearly.
6. The program includes example usage for demonstration. It makes the working of the system easy to understand for beginners.

## **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.

### **Code:**

```

class Student:
    """
    This class represents a student with a name and attendance record.

    Attributes:
    name (str): The name of the student.
    attendance (list): A list to store attendance records ('P' for present, 'A' for absent).
    """

    def __init__(self, name):
        self.name = name
        self.attendance = []

    def mark_attendance(self, status):
        """
        Marks the attendance for the student.

        Parameters:
        status (str): 'P' for present, 'A' for absent.
        """
        if status in ['P', 'A']:
            self.attendance.append(status)
        else:
            print("Invalid status. Use 'P' for present and 'A' for absent.")

    def display_attendance(self):
        """Displays the attendance record of the student."""
        print(f"Attendance record for {self.name}:")
        for day, status in enumerate(self.attendance, start=1):
            print(f"Day {day}: {status}")

# Example usage:
student_name = input("Enter the student's name: ")
student = Student(student_name)
days = int(input("Enter number of days to mark attendance: "))
for _ in range(days):
    status = input("Enter attendance status (P/A): ").strip().upper()
    student.mark_attendance(status)
student.display_attendance()

```

## Output:

```

PS C:\Users\madhu\OneDrive\Desktop\AI Assistant coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistant coding\Assignment(6.5)\Attendance.py"
Enter the student's name: madhu
Enter number of days to mark attendance: 5
Enter attendance status (P/A): p
Enter attendance status (P/A): p
Enter attendance status (P/A): p
Enter attendance status (P/A): a
Enter attendance status (P/A): a
Attendance record for madhu:
Day 1: P
Day 2: P
Day 3: P
Day 4: A
Day 5: A
PS C:\Users\madhu\OneDrive\Desktop\AI Assistant coding>

```

## Observation:

1. The program uses a Student class to store student details and attendance. This follows Object-Oriented Programming principles.
2. Attendance is stored in a list using P for present and A for absent. This makes attendance tracking simple and efficient.

3. Input validation is implemented for attendance status. It prevents invalid entries other than P and A.

4. Attendance is marked day by day using a loop. This allows flexible entry for any number of days.

5. The `display_attendance()` method shows attendance clearly with day numbers. It improves readability and understanding of records.

6. The example usage demonstrates real-time user input and output. This helps beginners easily understand how the program works.

### 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.

#Initialize ATM with a starting balance of \$100

**Code:**

```
class ATM:
    """
    This class simulates an ATM machine with basic functionalities like
    checking balance, depositing money, and withdrawing money.

    Attributes:
        balance (float): The current balance in the ATM account.
    """

    def __init__(self, initial_balance=0.0):
        self.balance = initial_balance

    def display_menu(self):
        """Displays the ATM menu options."""
        print("Welcome to the ATM!")
        print("1. Check Balance")
        print("2. Deposit Money")
        print("3. Withdraw Money")
        print("4. Exit")

    def check_balance(self):
        """Displays the current balance."""
        print(f"Your current balance is: ${self.balance:.2f}")

    def deposit_money(self, amount):
        """Deposits money into the account."""
        if amount > 0:
            self.balance += amount
            print(f"${amount:.2f} deposited successfully.")
        else:
            print("Deposit amount must be positive.")

    def withdraw_money(self, amount):
        """Withdraws money from the account if sufficient balance exists."""
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                print(f"${amount:.2f} withdrawn successfully.")
            else:
                print("Insufficient balance.")
        else:
            print("Withdrawal amount must be positive.")

    # Example usage:
    atm = ATM(100.0) # Initialize ATM with a starting balance of $100
    while True:
        atm.display_menu()
```

```

while True:
    atm.display_menu()
    choice = input("Please select an option (1-4): ")

    if choice == '1':
        atm.check_balance()
    elif choice == '2':
        amount = float(input("Enter amount to deposit: "))
        atm.deposit_money(amount)
    elif choice == '3':
        amount = float(input("Enter amount to withdraw: "))
        atm.withdraw_money(amount)
    elif choice == '4':
        print("Thank you for using the ATM. Goodbye!")
        break
    else:
        print("Invalid option. Please try again.")
print() # Print a newline for better readability between operations.

```

## Output:

```

PS C:\Users\madhu\OneDrive\Desktop\AI Assistant coding> python -u "c:\Users\madhu\OneDrive\Desktop\AI Assistant coding\Assignment(6.5)\atm.py"
Welcome to the ATM!
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Please select an option (1-4): 1
Your current balance is: $100.00

Welcome to the ATM!
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Please select an option (1-4): 2
Enter amount to deposit: 20
$20.00 deposited successfully.

Welcome to the ATM!
1. Check Balance
2. Deposit Money
3. Withdraw Money
4. Exit
Please select an option (1-4): 1
Your current balance is: $120.00

Welcome to the ATM!
1. Check Balance
2. Deposit Money
3. Withdraw Money

```

## Observation:

- 1.The program uses an ATM class to simulate basic ATM operations. This follows Object-Oriented Programming concepts.
- 2.The balance is stored as a class attribute and updated dynamically. It reflects deposits and withdrawals accurately.
- 3.A menu-driven approach is implemented using a loop. This allows the user to perform multiple operations continuously.
- 4.Input validation is done for deposit and withdrawal amounts. It prevents negative values and invalid transactions.

- 5.The program checks for sufficient balance before withdrawal. This avoids overdraft situations.
- 6.Clear and user-friendly messages are displayed for every operation. This improves usability and user understanding.