

Name: Achi Vaishnavi

Hall cket:2303A51554

Batch:29

Assessment: 6.5

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

Prompt:

Generate Python code to check voting eligibility based on age and citizenship.

Code:

```
# Generate Python code to check voting eligibility based on age and citizenship.
def check_voting_eligibility(age, is_citizen):
    if age >= 18 and is_citizen:
        return "Eligible to vote"
    else:
        return "Not eligible to vote"
age = 20
is_citizen = True
print(check_voting_eligibility(age, is_citizen))
age = 16
is_citizen = False
print(check_voting_eligibility(age, is_citizen))
```

Output:

```
• Eligible to vote
Not eligible to vote
```

Significance:

This task highlights the practical use of **conditional logic** in validating user information. It helps understand how multiple conditions are combined to make accurate decisions. Such logic is widely applied in online forms, authentication systems, and eligibility verification platforms.

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:

```
#Generate Python code to count vowels and consonants in a string using a loop.
def count_vowels_consonants(input_string):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0

    for char in input_string:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1

    return vowel_count, consonant_count
input_string = "Hello World"
vowels, consonants = count_vowels_consonants(input_string)
print(f"Vowels: {vowels}, Consonants: {consonants}")
```

Output:

```
Vowels: 3, Consonants: 7
```

Significance:

This task enhances knowledge of **loop-based string traversal**. It demonstrates how individual characters are processed and classified, forming the foundation for text analysis, input processing, and language-based applications.

Task 3: AI-Assisted Code Completion Reflection Task

Prompt:

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

Code:

```
#Generate a Python program for a library management system using classes, loops, and conditional statements.
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True
class Library:
    def __init__(self):
        self.books = []
    def add_book(self, book):
        self.books.append(book)
    def display_books(self):
        for book in self.books:
            status = "Available" if book.is_available else "Checked Out"
            print(f>Title: {book.title}, Author: {book.author}, Status: {status}")
    def check_out_book(self, title):
        for book in self.books:
            if book.title == title:
                if book.is_available:
                    book.is_available = False
                    print(f"You have checked out '{book.title}'.")
                    return
                else:
                    print(f"'{book.title}' is currently not available.")
                    return
        print(f"Book titled '{title}' not found in the library.")
    def return_book(self, title):
        for book in self.books:
            if book.title == title:
                book.is_available = True
                print(f"You have returned '{book.title}'.")
                return
        print(f"Book titled '{title}' not found in the library.")
library = Library()
library.add_book(Book("1984", "George Orwell"))
library.add_book(Book("To Kill a Mockingbird", "Harper Lee"))
library.add_book(Book("The Great Gatsby", "F. Scott Fitzgerald"))
library.display_books()
library.check_out_book("1984")
library.display_books()
library.return_book("1984")
library.display_books()
```

Output:

```

Title: 1984, Author: George Orwell, Status: Checked Out
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have checked out '1984'.
Title: 1984, Author: George Orwell, Status: Checked Out
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have checked out '1984'.
Title: 1984, Author: George Orwell, Status: Checked Out
You have checked out '1984'.
Title: 1984, Author: George Orwell, Status: Checked Out
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: 1984, Author: George Orwell, Status: Checked Out
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available
You have returned '1984'.
Title: 1984, Author: George Orwell, Status: Available
Title: To Kill a Mockingbird, Author: Harper Lee, Status: Available
Title: The Great Gatsby, Author: F. Scott Fitzgerald, Status: Available

```

Significance:

This task provides hands-on experience with **object-oriented programming concepts**. It shows how classes can organize data and behavior efficiently. This approach is essential for building scalable systems like library software, inventory management, and student databases.

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

Prompt:

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

Code:

```

#Generate a Python class to mark and display student attendance using loops.
class Student:
    def __init__(self, name):
        self.name = name
        self.attendance = []
    def mark_attendance(self, day):
        self.attendance.append(day)
    def display_attendance(self):
        print(f"Attendance for {self.name}: {' '.join(self.attendance)}")
student1 = Student("Alice")
student2 = Student("Bob")
student1.mark_attendance("Monday")
student1.mark_attendance("Wednesday")
student2.mark_attendance("Tuesday")
student2.mark_attendance("Thursday")
student1.display_attendance()
student2.display_attendance()

```

Output:

```
Attendance for Alice: Monday, Wednesday
Attendance for Bob: Tuesday, Thursday
```

Significance:

This task focuses on **structured data storage and retrieval** using classes and loops. It represents real-world academic automation systems and helps develop skills in managing records programmatically.

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

Prompt:

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

Code:

```
#Generate a Python program using loops and conditionals to simulate an ATM menu.
class ATM:
    def __init__(self, balance=0):
        self.balance = balance
    def display_menu(self):
        print("ATM Menu:")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")
        print("4. Exit")
    def check_balance(self):
        print(f"Your current balance is: ${self.balance}")
    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"You have deposited: ${amount}")
        else:
            print("Invalid deposit amount.")
    def withdraw(self, amount):
        if amount > self.balance:
            print("Insufficient funds.")
        elif amount <= 0:
            print("Invalid withdrawal amount.")
        else:
            self.balance -= amount
            print(f"You have withdrawn: ${amount}")
atm = ATM(1000)
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 the amount to deposit: "))
        atm.deposit(amount)
    elif choice == '3':
        amount = float(input("Enter the amount to withdraw: "))
        atm.withdraw(amount)
    elif choice == '4':
        print("Thank you for using the ATM. Goodbye!")
        break
    else:
        print("Invalid option. Please try again.")
```

Output:

```
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 1
Your current balance is: $1000
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 2
Enter the amount to deposit: 2000
You have deposited: $2000.0
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 3
Enter the amount to withdraw: 1000
You have withdrawn: $1000.0
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 4
2. Deposit
3. Withdraw
4. Exit
Please select an option (1-4): 4
Thank you for using the ATM. Goodbye!
```

Significance:

This task demonstrates **interactive menu-driven programming**. It teaches how user choices are handled dynamically using loops and conditionals, which is fundamental in financial applications, kiosks, and service-based systems.