ASSIGNMENT:06.5

HALLTICKET:2303A51598

BATCH:29

SHAIK NEHA FARIYAL

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.

```python
#Generate Python code to check voting eligibility based on age and citizenship.
def check_voting_eligibility(age, citizenship):
    """Check if a person is eligible to vote based on age and citizenship."""
    if age >= 18 and citizenship.lower() == "yes":
        return "Eligible to vote."
    elif age < 18:
        return "Not eligible to vote: Age must be at least 18."
    else:
        return "Not eligible to vote: Citizenship is required."
# Example usage
age_input = 20
citizenship_input = "Yes"
eligibility_result = check_voting_eligibility(age_input, citizenship_input)
print(eligibility_result)
```

OUTPUT:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Eligible to vote.
PS C:\Users\nehaf>
```

JUSTIFICATION:

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 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.

```
#Generate Python code to count vowels and consonants in a string using a loop.
def count_vowels_consonants(input_string):
    """Count the number of vowels and consonants in a given string."""
    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
# Example usage
input_str = "Hello, World!"
vowels, consonants = count_vowels_consonants(input_str)
print(f"In the string '{input_str}', there are {vowels} vowels and {consonants} consonants.")
```

OUTPUT:

```
PS C:\Users\nehaf\OneDrive\Desktop\java files> & 'c:\Users\nehaf\AppData\Local\Programs\Python\Python313\python.exe' 'c:\Users\nehaf\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '54914' '--' 'C:\Users\nehaf\OneDrive\Desktop\java files\lab1.py'
In the string 'Hello, World!', there are 3 vowels and 7 consonants.
PS C:\Users\nehaf\OneDrive\Desktop\java files>
```

JUSTIFICATION:

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 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

CODE:

```python
#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)
        print(f"Added '{book.title}' by {book.author} to the library.")

    def borrow_book(self, title):
        for book in self.books:
            if book.title == title:
                if book.is_available:
                    book.is_available = False
                    print(f"You have borrowed '{book.title}'.")
                    return
                else:
                    print(f"Sorry, '{book.title}' is currently unavailable.")
                    return
        print(f"Sorry, '{title}' is not found in the library.")

    def return_book(self, title):
        for book in self.books:
            if book.title == title:
                if not book.is_available:
                    book.is_available = True
                    print(f"You have returned '{book.title}'.")
```

```
                    return
                else:
                    print(f"'{book.title}' was not borrowed.")
                    return
        print(f"Sorry, '{title}' is not found in the library.")
# Example usage
library = Library()
book1 = Book("To Kill a Mockingbird", "Harper Lee")
book2 = Book("1984", "George Orwell")
library.add_book(book1)
library.add_book(book2)
library.borrow_book("1984")
library.borrow_book("1984")
library.return_book("1984")
library.return_book("1984")
library.borrow_book("The Great Gatsby")

```

Output:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Added 'To Kill a Mockingbird' by Harper Lee to the library.
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Added 'To Kill a Mockingbird' by Harper Lee to the library.
Added 'To Kill a Mockingbird' by Harper Lee to the library.
Added '1984' by George Orwell to the library.
You have borrowed '1984'.
Sorry, '1984' is currently unavailable.
You have returned '1984'.
Sorry, '1984' is currently unavailable.
You have returned '1984'.
You have returned '1984'.
'1984' was not borrowed.
Sorry, 'The Great Gatsby' is not found in the library.
PS C:\Users\nehaf>
```

JUSTIFICATION:

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 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.

```python
#Generate a Python class to mark and display student attendance using loops.
class Student:
    def __init__(self, name):
        self.name = name
class Attendance:
    def __init__(self):
        self.attendance_record = {}
    def mark_attendance(self, student):
        self.attendance_record[student.name] = "Present"
        print(f"{student.name} marked as present.")
    def display_attendance(self):
        print("Attendance Record:")
        for name, status in self.attendance_record.items():
            print(f"{name}: {status}")
# Example usage
student1 = Student("Alice")
student2 = Student("Bob")
attendance = Attendance()
attendance.mark_attendance(student1)
attendance.mark_attendance(student2)
attendance.display_attendance()
```

OUTPUT:

```
PS C:\Users\nehaf> & C:/Users/nehaf/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/nehaf/OneDrive/Desktop/java files/lab1.py"
Alice marked as present.
Bob marked as present.
Attendance Record:
Alice: Present
Bob: Present
PS C:\Users\nehaf>
```

JUSTIFICATIONS:

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 Description #5 (AI-Based Code Completion for Conditional Menu Navigation)

PROMPT:

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

```python
#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("\nATM Menu:")
        print("1. Check Balance")
        print("2. Deposit")
        print("3. Withdraw")

    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:
            self.balance -= amount
            print(f"You have withdrawn: ${amount}")
        else:
            print("Invalid withdrawal amount.")
# Example usage
atm = ATM(100)  # Initialize ATM with a balance of $100
while True:
    atm.display_menu()
    choice = input("Please select an option (1-3) or 'q' to quit: ")
    if choice == '1':
        atm.check_balance()
    elif choice == '2':
```

```python
        elif amount > 0:
            self.balance -= amount
            print(f"You have withdrawn: ${amount}")
        else:
            print("Invalid withdrawal amount.")
# Example usage
atm = ATM(100)  # Initialize ATM with a balance of $100
while True:
    atm.display_menu()
    choice = input("Please select an option (1-3) or 'q' to quit: ")
    if choice == '1':
        atm.check_balance()
    elif choice == '2':
        amount = float(input("Enter deposit amount: "))
        atm.deposit(amount)
    elif choice == '3':
        amount = float(input("Enter withdrawal amount: "))
        atm.withdraw(amount)
    elif choice.lower() == 'q':
        print("Thank you for using the ATM. Goodbye!")
        break
    else:
        print("Invalid option. Please try again.")
```

OUTPUT:

JUSTIFICATION:

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.