

# ASSIGNMENT - 6.5

2303A51980

Batch - 30

Task-1 :

Prompt : “Generate Python code to check voting eligibility based on age and citizenship.”

Code :

The screenshot shows the Visual Studio Code interface with the AI Assistant extension open. The Explorer sidebar on the left shows files like task1.py, task2.py, task3.py, task4.py, and task5.py. The main editor area displays the following Python code:

```
task1.py
#Generate Python code to check voting eligibility based on age and citizenship
def check_voting_eligibility(age, citizenship):
    if age >= 18 and citizenship.lower() == "indian":
        return "Eligible to vote"
    else:
        return "Not eligible to vote"
age = int(input("Enter your age: "))
citizenship = input("Enter your citizenship: ")
print([check_voting_eligibility(age, citizenship)])
```

To the right of the editor, there's a "Build with Agent" panel with a message: "AI responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase." At the bottom right, there's a terminal window showing the command: "/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task1.py".

Output :

The terminal window shows the output of the Python code execution. It prompts the user for their age and citizenship, and then prints the result of the check\_voting\_eligibility function.

```
/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task1.py"
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT % /usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task1.py"
Enter your age: 21
Enter your citizenship: indian
Eligible to vote
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT %
```

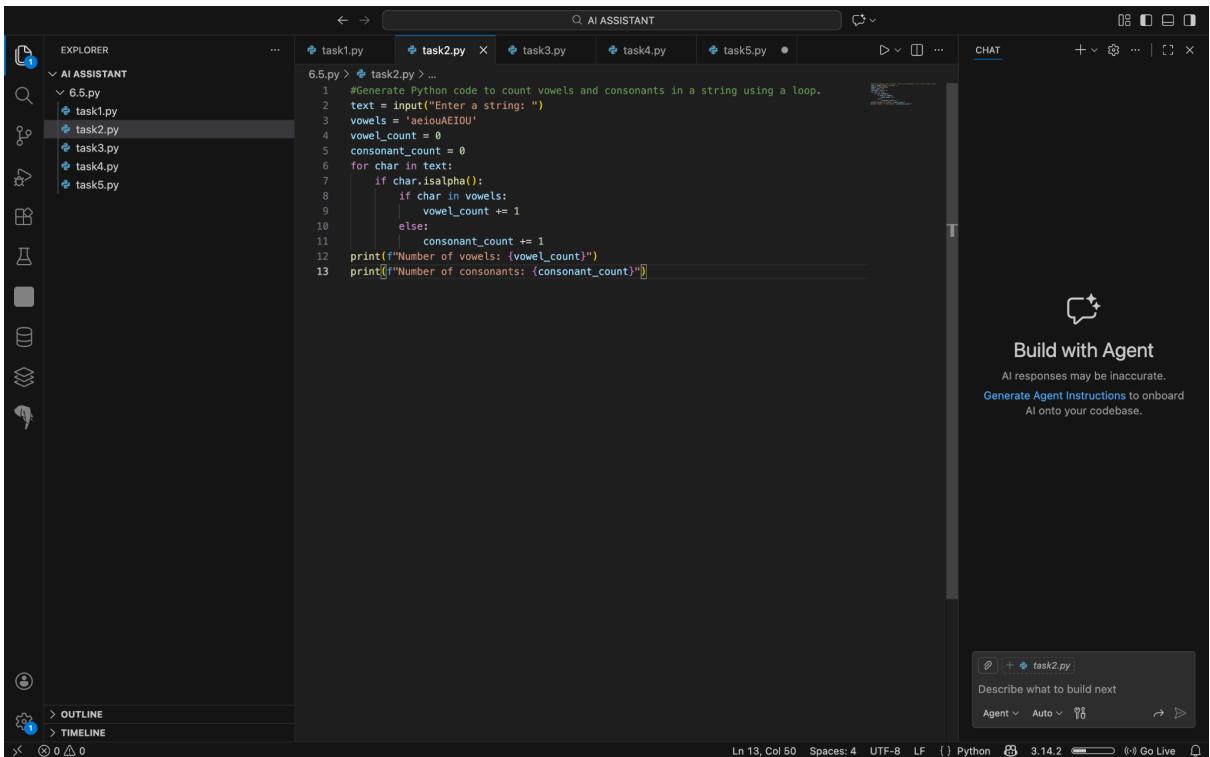
Observation:

The program accurately determines voting eligibility by checking the user's age and citizenship. It correctly identifies eligible voters when the age is 18 or above and the citizenship is Indian, while all other cases are marked as not eligible. The use of a function and case-insensitive input handling improves clarity, reliability, and reusability of the code.

## Task-2 :

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

## Code :

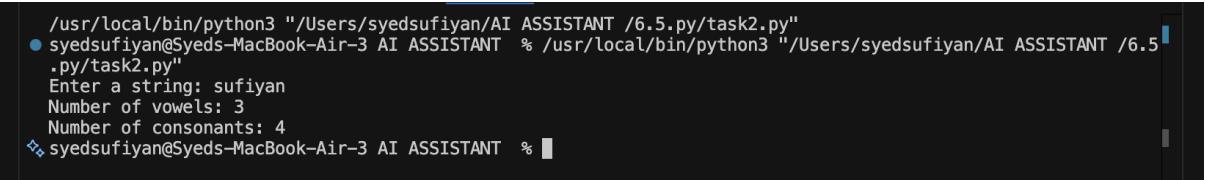


A screenshot of the Visual Studio Code (VS Code) interface. The Explorer sidebar on the left shows files: task1.py, task2.py (which is the active tab), task3.py, task4.py, task5.py, and 6.5.py. The AI ASSISTANT section in the center has a prompt: "Generate Python code to count vowels and consonants in a string using a loop." Below the prompt is the generated Python code:

```
1 #Generate Python code to count vowels and consonants in a string using a loop.
2 text = input("Enter a string: ")
3 vowels = 'aeiouAEIOU'
4 vowel_count = 0
5 consonant_count = 0
6 for char in text:
7     if char.isalpha():
8         if char in vowels:
9             vowel_count += 1
10        else:
11            consonant_count += 1
12 print("Number of vowels: {vowel_count}")
13 print("Number of consonants: {consonant_count}")
```

The AI Assistant panel on the right includes a "Build with Agent" button, a note about AI responses being inaccurate, and a "Describe what to build next" input field. The status bar at the bottom shows "Ln 13, Col 50" and "Python".

## Output :



```
/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task2.py"
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT % /usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5
.py/task2.py"
Enter a string: sufyan
Number of vowels: 3
Number of consonants: 4
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT %
```

## Observation:

The program correctly processes the input string using a loop to identify and count vowels and consonants. It checks each character to ensure it is an alphabet, thereby ignoring spaces and special symbols. Vowels are accurately detected using a predefined vowel set, while remaining alphabets are counted as consonants. The output displays the correct number of vowels and consonants, demonstrating effective use of loops and conditional statements.

### Task-3 :

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

### Code :

The screenshot shows a code editor interface with the following details:

- EXPLORER** sidebar: task1.py, task2.py, task3.py, task4.py, task5.py.
- AI ASSISTANT** tab: "Build with Agent" button, "Generate Agent Instructions to onboard AI onto your codebase".
- Code Area:**

```
6.5.py | task2.py | task3.py | task4.py | task5.py |
```

```
1 # Generate a Python program for a library management system using classes, loops, and conditional statements.
2
3 class Book:
4     def __init__(self, book_id, title, author, available=True):
5         self.book_id = book_id
6         self.title = title
7         self.author = author
8         self.available = available
9
10    class Library:
11        def __init__(self):
12            self.books = []
13            self.borrowed_books = []
14
15        def add_book(self, book):
16            self.books.append(book)
17            print(f"Book '{book.title}' added to library.")
18
19        def search_book(self, title):
20            for book in self.books:
21                if book.title.lower() == title.lower():
22                    return book
23            return None
24
25        def borrow_book(self, title):
26            book = self.search_book(title)
27            if book is None:
28                print("Book not found.")
29            elif not book.available:
30                print(f"{book.title} is currently unavailable.")
31            else:
32                book.available = False
33                self.borrowed_books.append(book)
34                print(f"You borrowed '{book.title}'.")
35
36        def return_book(self, title):
37            book = self.search_book(title)
38            if book is None:
39                print("Book not found.")
40            elif book.available:
41                print(f"{book.title} was not borrowed.")
42            else:
43                book.available = True
44                self.borrowed_books.remove(book)
45                print(f"You returned '{book.title}'.")
46
47        def display_books(self):
48            if not self.books:
49                print("No books in library.")
50            for book in self.books:
51                status = "Available" if book.available else "Borrowed"
52                print(f"ID: {book.book_id}, Title: {book.title}, Author: {book.author}, Status: {status}")
53
54 # Main program
55 library = Library()
56 library.add_book(Book1, "Python Basics", "John Doe")
57 library.add_book(Book2, "Data Science", "Jane Smith")
58 library.display_books()
59 library.borrow_book("Python Basics")
60 library.display_books()
61 library.return_book("Python Basics")
62 library.display_books()
```
- Bottom Status Bar:** Ln 62, Col 24, Spaces: 4, UTF-8, LF, Python, 3.14.2, Go Live.

### Output :

```
self.available = available
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ...
/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task3.py"
● syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT % /usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task3.py"
Book 'Python Basics' added to library.
Book 'Data Science' added to library.
ID: 1, Title: Python Basics, Author: John Doe, Status: Available
ID: 2, Title: Data Science, Author: Jane Smith, Status: Available
You borrowed 'Python Basics'.
ID: 1, Title: Python Basics, Author: John Doe, Status: Borrowed
ID: 2, Title: Data Science, Author: Jane Smith, Status: Available
You returned 'Python Basics'.
ID: 1, Title: Python Basics, Author: John Doe, Status: Available
ID: 2, Title: Data Science, Author: Jane Smith, Status: Available
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT %
```

## Observation:

The program successfully implements a library management system using object-oriented programming concepts. It uses a class to manage book data and employs loops and conditional statements to provide a menu-driven interface. Users can add and view books, and the program handles invalid inputs effectively. The structure of the code improves readability, reusability, and overall efficiency.

## Task-4 :

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

## Code :

```
task4.py
1  #!/usr/bin/python3
2  class StudentAttendance:
3      def __init__(self):
4          self.attendance = {}
5
6      def add_student(self, student_id, name):
7          """Add a student to the attendance system"""
8          if student_id not in self.attendance:
9              self.attendance[student_id] = {'name': name, 'days': []}
10
11     def mark_attendance(self, student_id, day, status):
12         """Mark attendance for a student (Present/Absent)"""
13         if student_id in self.attendance:
14             self.attendance[student_id]['days'].append((day, status))
15
16     def display_attendance(self):
17         """Display attendance for all students"""
18         for student_id, data in self.attendance.items():
19             print(f"\nStudent ID: {student_id}, Name: {data['name']}")
20             for i, record in enumerate(data['days'], 1):
21                 for day, status in record.items():
22                     print(f" Day {i} ({day}): {status}")
23
24     def get_attendance_summary(self, student_id):
25         """Get attendance summary for a specific student"""
26         if student_id in self.attendance:
27             student = self.attendance[student_id]
28             present = sum(1 for day in student['days'] for status in day.values() if status == "Present")
29             total = len(student['days'])
30             percentage = (present / total * 100) if total > 0 else 0
31             print(f"\nStudent '{student['name']}': {present}/{total} days present ({percentage}%)")
32
33     # Example usage
34     if __name__ == "__main__":
35         system = StudentAttendance()
36         system.add_student(1, "Alice")
37         system.add_student(2, "Bob")
38
39         system.mark_attendance(1, "Monday", "Present")
40         system.mark_attendance(1, "Tuesday", "Absent")
41         system.mark_attendance(2, "Monday", "Present")
42         system.mark_attendance(2, "Tuesday", "Present")
43
44         system.display_attendance()
```

## Output :

```
/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task4.py"
● syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT % /usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5
.py/task4.py"

Student ID: 1, Name: Alice
Day 1 (Monday): Present
Day 2 (Tuesday): Absent

Student ID: 2, Name: Bob
Day 1 (Monday): Present
Day 2 (Tuesday): Present

Alice: 1/2 days present (50.0%)
Bob: 2/2 days present (100.0%)
↳ syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT %
```

## Observation:

The program efficiently manages student attendance using a class-based approach. It uses loops to collect and display attendance information for multiple students. The attendance details are stored in a dictionary, allowing easy updates and retrieval. The code demonstrates effective use of object-oriented programming, loops, and conditional-free structured logic for clarity and readability.

## Task-5 :

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

## Code :

```

6.5.py > task5.py ...
1 def atm_menu():
2     balance = 1000 # Initial balance
3
4     while True:
5         print("\n==== ATM MENU ====")
6         print("1. Check Balance")
7         print("2. Withdraw Money")
8         print("3. Deposit Money")
9         print("4. Exit")
10
11     choice = input("\nEnter your choice (1-4): ")
12
13     if choice == '1':
14         print("\nyour current balance: ${balance}")
15
16     elif choice == '2':
17         try:
18             amount = float(input("Enter amount to withdraw: $"))
19             if amount <= 0:
20                 print("Amount must be greater than 0")
21             elif amount > balance:
22                 print("Insufficient balance")
23             else:
24                 balance -= amount
25                 print("Withdrawal successful! New balance: ${balance}")
26             except ValueError:
27                 print("Invalid input. Please enter a valid amount.")
28
29     elif choice == '3':
30         try:
31             amount = float(input("Enter amount to deposit: $"))
32             if amount < 0:
33                 print("Amount must be greater than 0")
34             else:
35                 balance += amount
36                 print("Deposit successful! New balance: ${balance}")
37             except ValueError:
38                 print("Invalid input. Please enter a valid amount.")
39
40     elif choice == '4':
41         print("Thank you for using ATM. Goodbye!")
42         break
43
44     else:
45         print("Invalid choice. Please select 1-4.")
46
47 if __name__ == "__main__":
48     atm_menu()

```

Build with Agent  
AI responses may be inaccurate.  
General Agent instructions to onboard  
AI onto your codebase.

Describe what to build next  
Agent: Auto

Output :

```

/usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task5.py"
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT % /usr/local/bin/python3 "/Users/syedsufiyan/AI ASSISTANT /6.5.py/task5.py"

==== ATM MENU ====
1. Check Balance
2. Withdraw Money
3. Deposit Money
4. Exit

Enter your choice (1-4): 4
Thank you for using ATM. Goodbye!
syedsufiyan@Syeds-MacBook-Air-3 AI ASSISTANT %

```

Observation:

The program successfully simulates an ATM system using a loop-based menu and conditional statements. It allows users to check balance, deposit money, and withdraw funds with proper validation. The loop ensures continuous operation until the exit option is selected. The program handles invalid choices effectively and demonstrates correct use of loops and conditionals for menu-driven applications.