Data Structure Assignment- 1
Course Code: ENCS205

TITLE:
Weather Data Storage using Arrays (Python Implementation)

Name: Simran
Roll no. 2401420052
Semester: 3rd, B. Tech CSE Data Science

INTRODUCTION

This project implements a simple weather data storage system using python.
The system uses a 2D array (rows= years, columns= cities) to store sparse temperature data.
Each entry represents the temperature of a particular city in a given year.
A sentinel value (-999) is used to represent missing data.

This project also demonstrates basic operations such as insertion, deletion, retrieval, and traversal of data. Additionally, it highlights the concepts of row-major and column major access in arrays, which are important for understanding how data is stored and accessed in memory.

OBJECTIVES

- To implement a Weather Data Storage System using arrays in python.
- To learn how 2D arrays can represent real-world tabular data (years * cities).
- To perform basic operations: Insert, Delete, and Retrieve weather records.
- To understand row-major and column major access in arrays.
- To handle sparse data using sentinel value (-999).

SYSTEM DESIGN

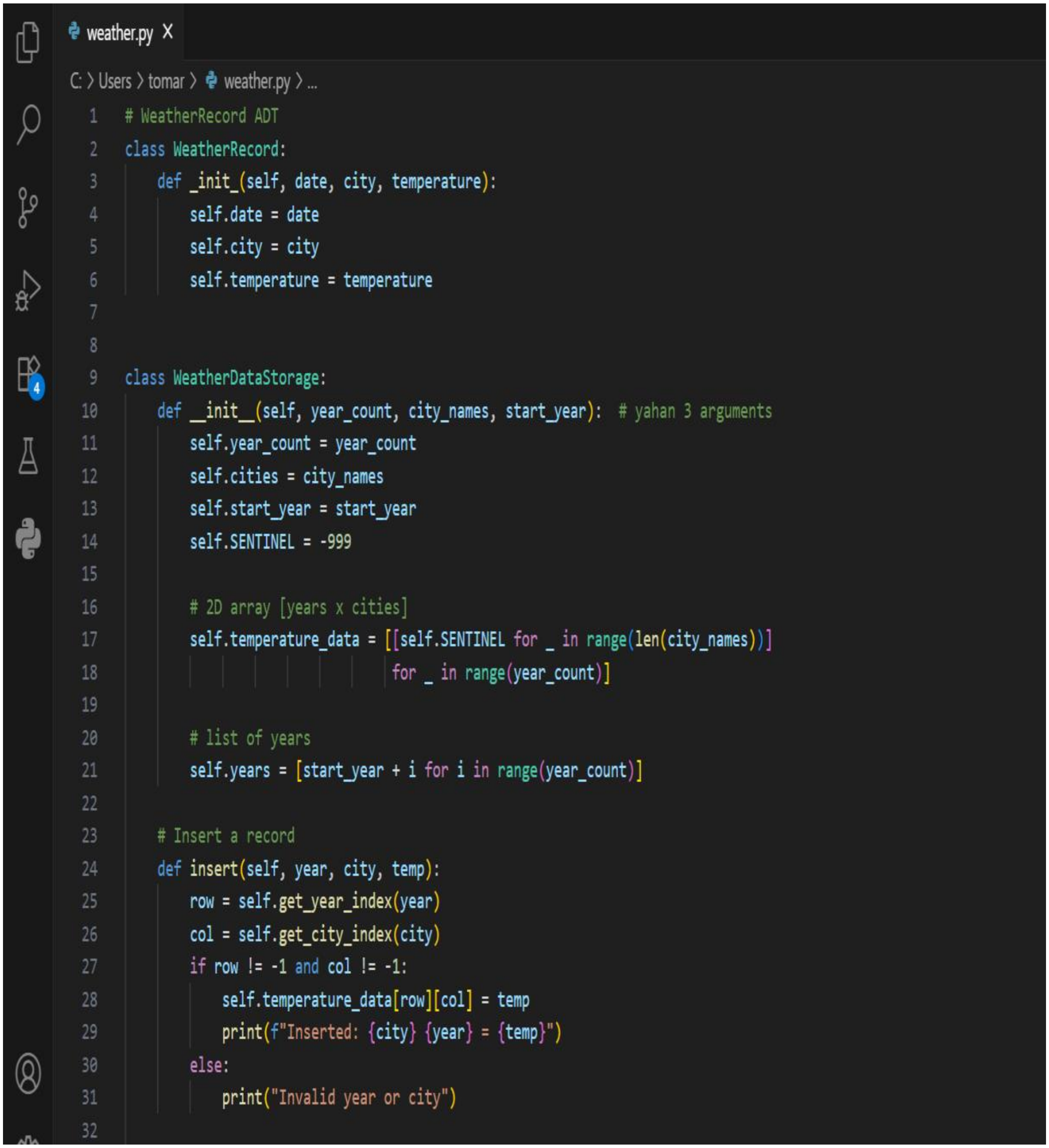
Classes Implemented:

1. _WeatherRecord: Stores individual weather data entries (date, city, temperature).
2. WeatherDataStorage: Implements a 2D array for years*cities and manages operations.

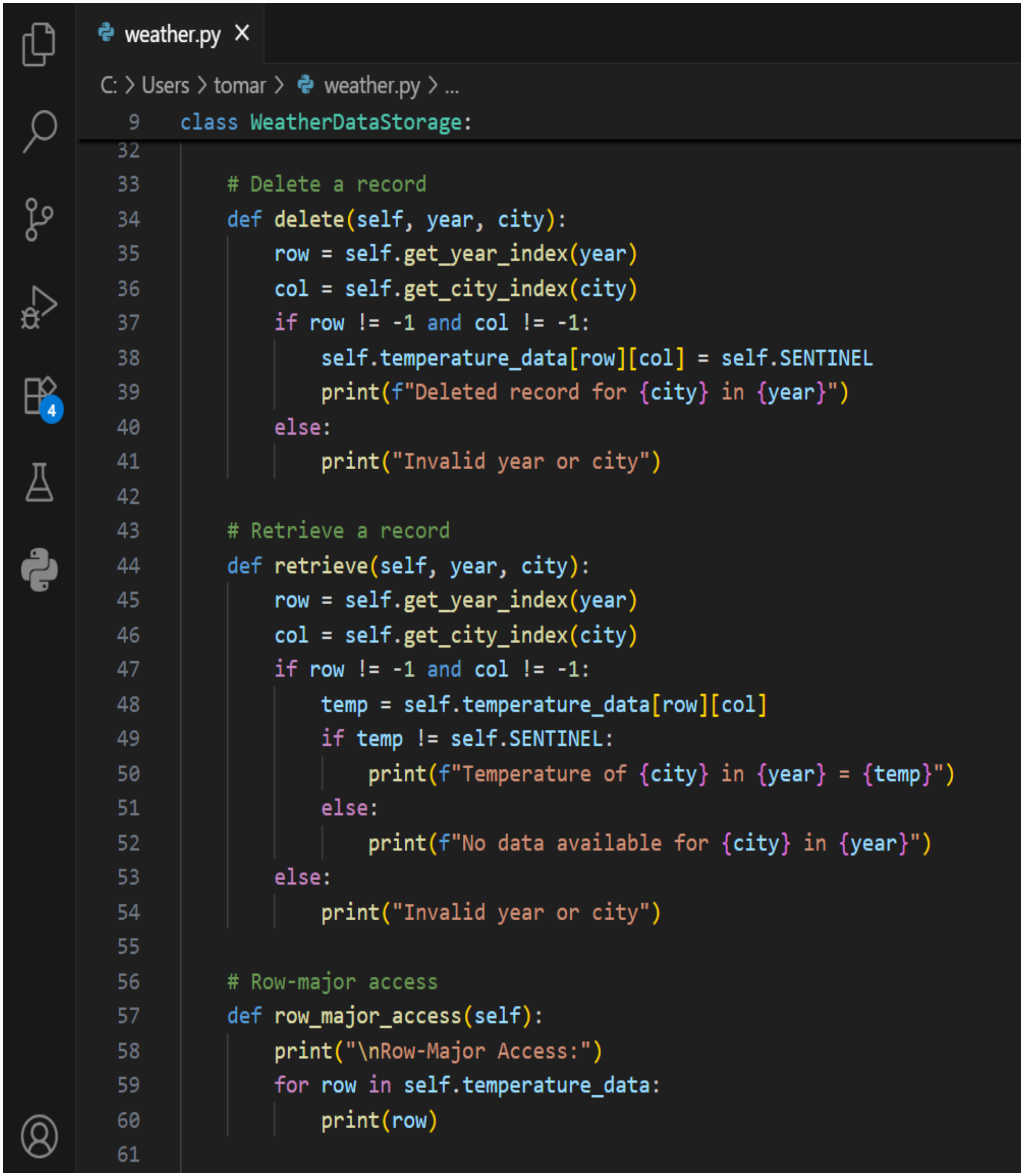
Operations Supported:

- insert(year, city, temp): Store temperature
- delete(year, city): Remove a record (replace with sentinel).
- retrieve(year, city): Fetch temperature for a given city & year.
- row_major_access(): print data row by row.
- column_major_access(): print data column by column.

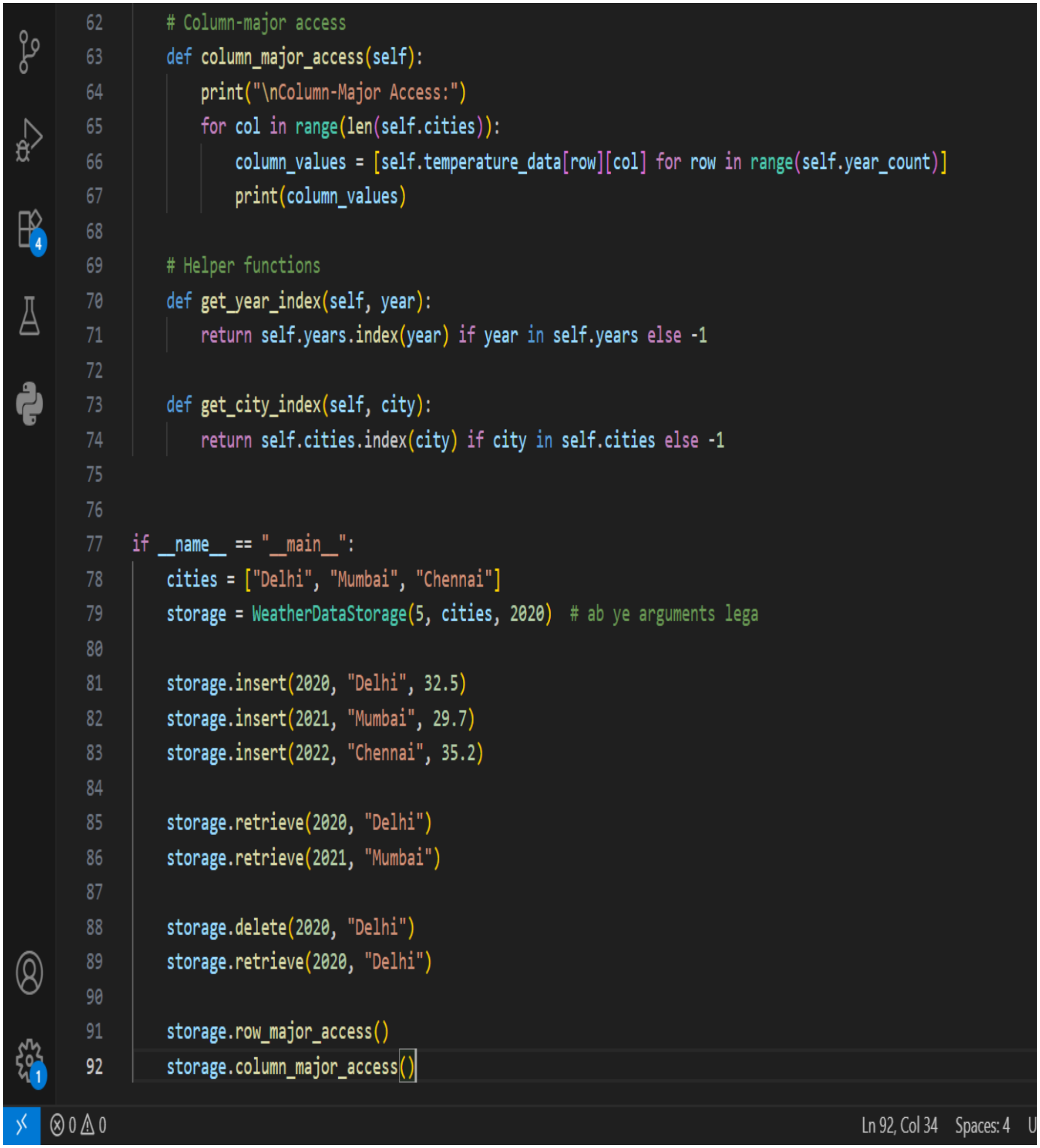
IMPLEMENTATION



```python
# WeatherRecord ADT
class WeatherRecord:
    def _init_(self, date, city, temperature):
        self.date = date
        self.city = city
        self.temperature = temperature


class WeatherDataStorage:
    def __init__(self, year_count, city_names, start_year):  # yahan 3 arguments
        self.year_count = year_count
        self.cities = city_names
        self.start_year = start_year
        self.SENTINEL = -999

        # 2D array [years x cities]
        self.temperature_data = [[self.SENTINEL for _ in range(len(city_names))]
                                 for _ in range(year_count)]

        # list of years
        self.years = [start_year + i for i in range(year_count)]

    # Insert a record
    def insert(self, year, city, temp):
        row = self.get_year_index(year)
        col = self.get_city_index(city)
        if row != -1 and col != -1:
            self.temperature_data[row][col] = temp
            print(f"Inserted: {city} {year} = {temp}")
        else:
            print("Invalid year or city")

```

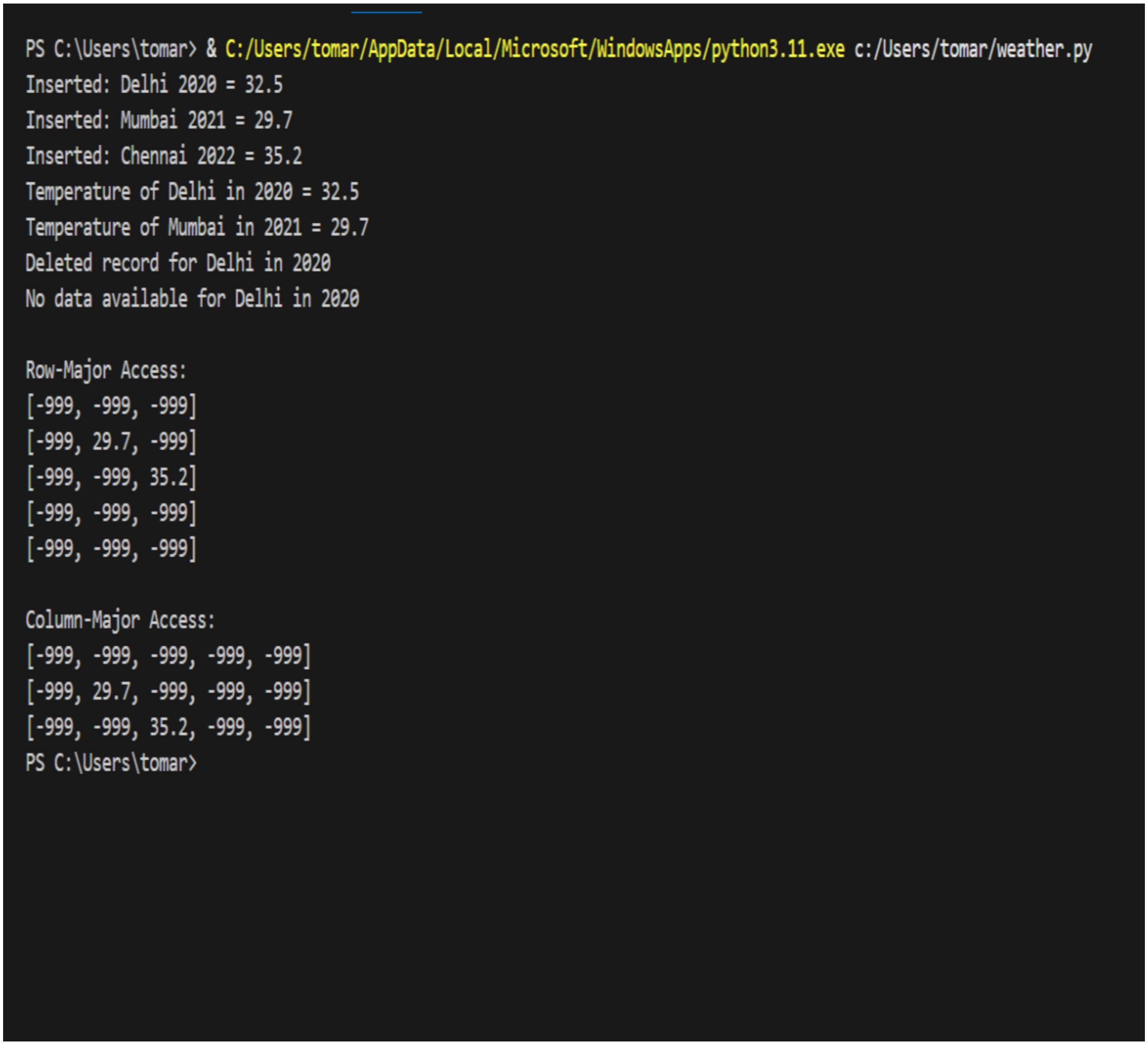


```python
class WeatherDataStorage:

    # Delete a record
    def delete(self, year, city):
        row = self.get_year_index(year)
        col = self.get_city_index(city)
        if row != -1 and col != -1:
            self.temperature_data[row][col] = self.SENTINEL
            print(f"Deleted record for {city} in {year}")
        else:
            print("Invalid year or city")

    # Retrieve a record
    def retrieve(self, year, city):
        row = self.get_year_index(year)
        col = self.get_city_index(city)
        if row != -1 and col != -1:
            temp = self.temperature_data[row][col]
            if temp != self.SENTINEL:
                print(f"Temperature of {city} in {year} = {temp}")
            else:
                print(f"No data available for {city} in {year}")
        else:
            print("Invalid year or city")

    # Row-major access
    def row_major_access(self):
        print("\nRow-Major Access:")
        for row in self.temperature_data:
            print(row)

```



```
    # Column-major access
    def column_major_access(self):
        print("\nColumn-Major Access:")
        for col in range(len(self.cities)):
            column_values = [self.temperature_data[row][col] for row in range(self.year_count)]
            print(column_values)

    # Helper functions
    def get_year_index(self, year):
        return self.years.index(year) if year in self.years else -1

    def get_city_index(self, city):
        return self.cities.index(city) if city in self.cities else -1


if __name__ == "__main__":
    cities = ["Delhi", "Mumbai", "Chennai"]
    storage = WeatherDataStorage(5, cities, 2020)  # ab ye arguments lega

    storage.insert(2020, "Delhi", 32.5)
    storage.insert(2021, "Mumbai", 29.7)
    storage.insert(2022, "Chennai", 35.2)

    storage.retrieve(2020, "Delhi")
    storage.retrieve(2021, "Mumbai")

    storage.delete(2020, "Delhi")
    storage.retrieve(2020, "Delhi")

    storage.row_major_access()
    storage.column_major_access()
```

OUTPUT



```
PS C:\Users\tomar> & C:/Users/tomar/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/tomar/weather.py
Inserted: Delhi 2020 = 32.5
Inserted: Mumbai 2021 = 29.7
Inserted: Chennai 2022 = 35.2
Temperature of Delhi in 2020 = 32.5
Temperature of Mumbai in 2021 = 29.7
Deleted record for Delhi in 2020
No data available for Delhi in 2020

Row-Major Access:
[-999, -999, -999]
[-999, 29.7, -999]
[-999, -999, 35.2]
[-999, -999, -999]
[-999, -999, -999]

Column-Major Access:
[-999, -999, -999, -999, -999]
[-999, 29.7, -999, -999, -999]
[-999, -999, 35.2, -999, -999]
PS C:\Users\tomar>
```

COMPLEXITY ANALYSIS

- Insert: O(1) (direct indexing by year & city)
- Delete: O(1)
- Retrieve: O(1)
- Row-Major Access: O(N × M), where N = years, M = cities
- Column-Major Access: O(N × M)

CONCLUSION

This project successfully demonstrates how a 2D array can be used to implement a simple Weather Data Storage ADT.

The system supports insertion, deletion, retrieval, and traversal (row-major and column-major) efficiently.

It provides hands-on practice of array data structure concepts and shows how arrays can be applied to a real-world scenario like weather monitoring.