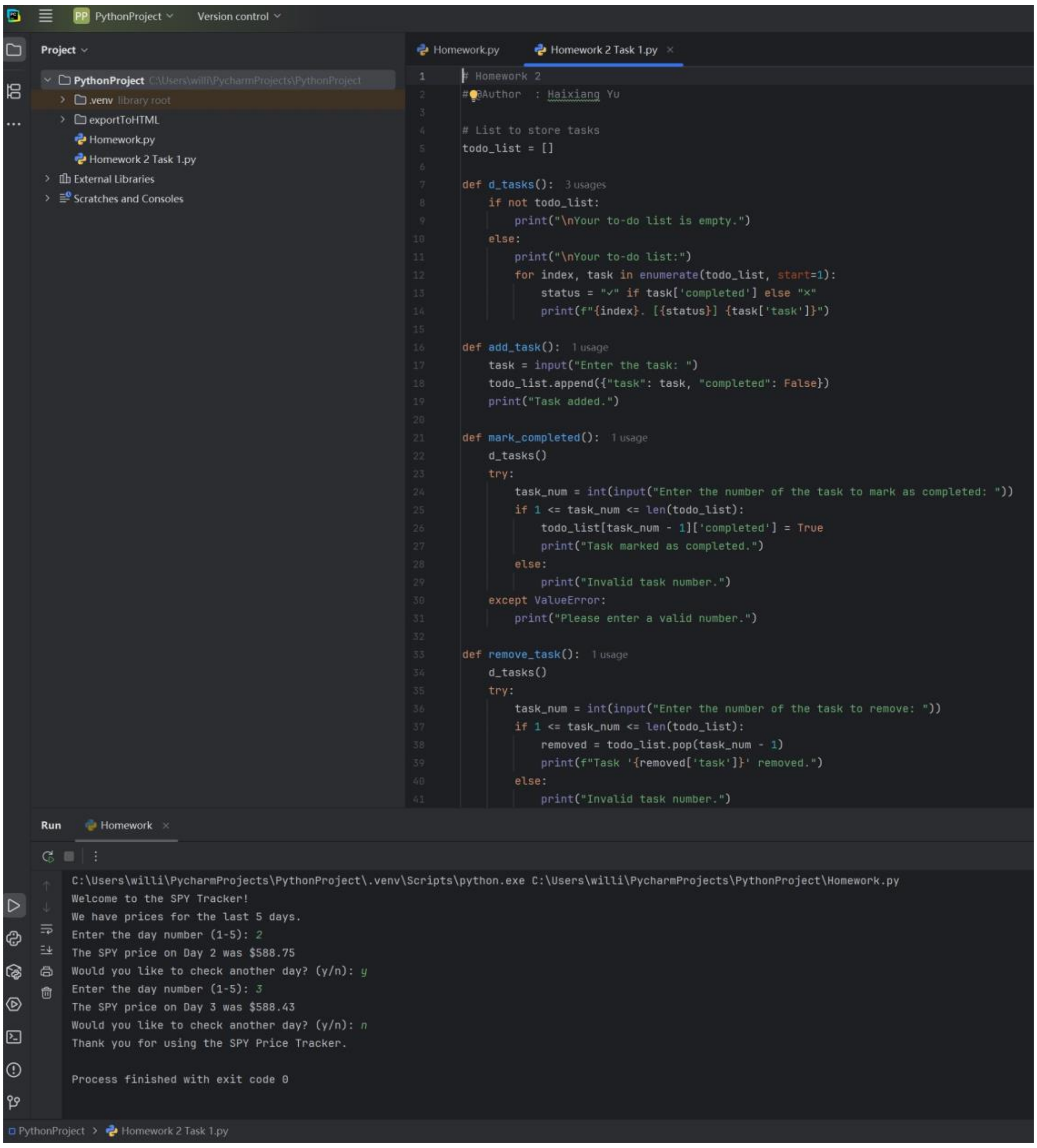
Haixiang Yu

PRG 501

Homework 2 Part 1

Task 1

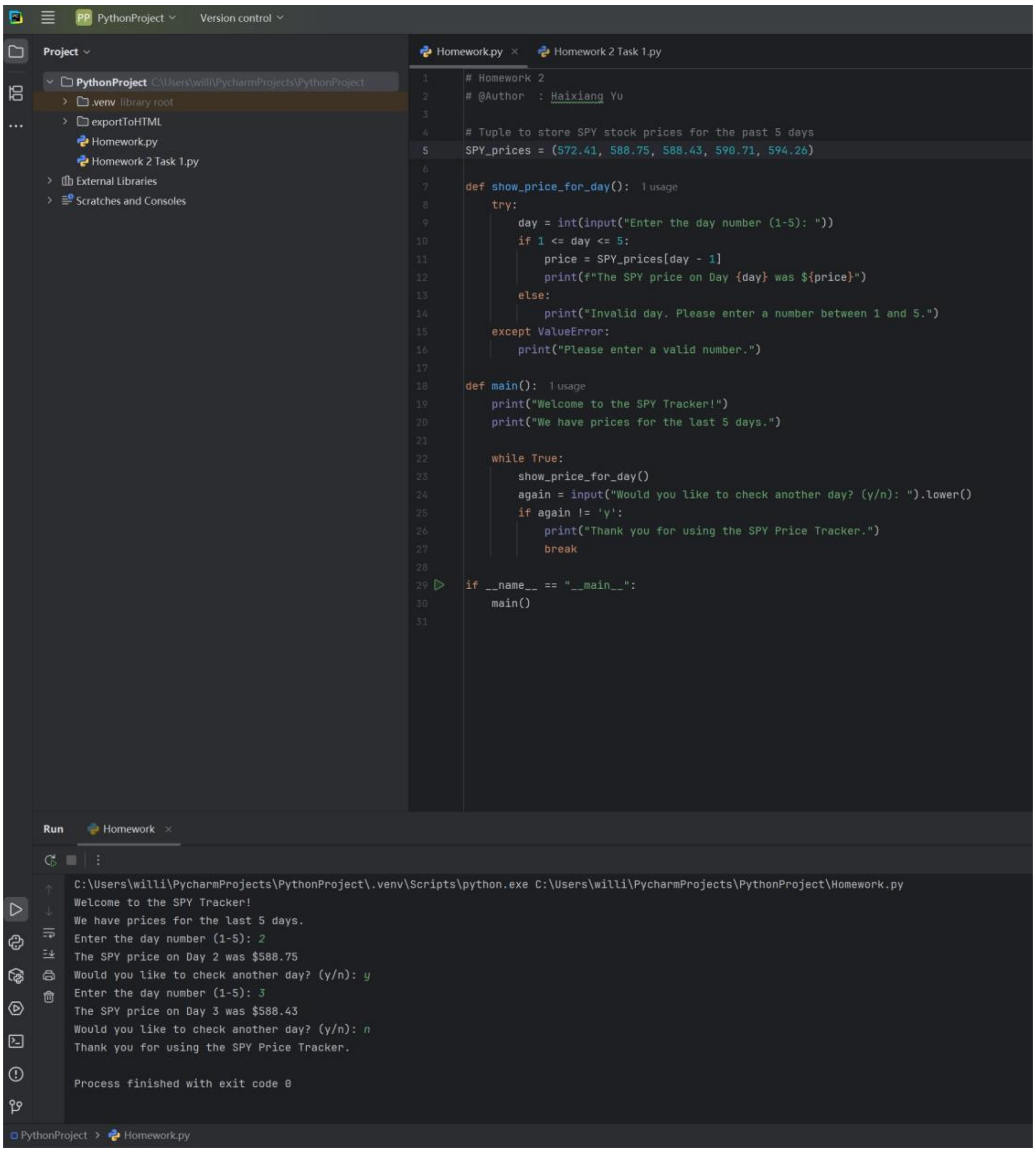


```python
# Homework 2
#@Author  : Haixiang Yu

# List to store tasks
todo_list = []

def d_tasks():
    if not todo_list:
        print("\nYour to-do list is empty.")
    else:
        print("\nYour to-do list:")
        for index, task in enumerate(todo_list, start=1):
            status = "✓" if task['completed'] else "×"
            print(f"{index}. [{status}] {task['task']}")

def add_task():
    task = input("Enter the task: ")
    todo_list.append({"task": task, "completed": False})
    print("Task added.")

def mark_completed():
    d_tasks()
    try:
        task_num = int(input("Enter the number of the task to mark as completed: "))
        if 1 <= task_num <= len(todo_list):
            todo_list[task_num - 1]['completed'] = True
            print("Task marked as completed.")
        else:
            print("Invalid task number.")
    except ValueError:
        print("Please enter a valid number.")

def remove_task():
    d_tasks()
    try:
        task_num = int(input("Enter the number of the task to remove: "))
        if 1 <= task_num <= len(todo_list):
            removed = todo_list.pop(task_num - 1)
            print(f"Task '{removed['task']}' removed.")
        else:
            print("Invalid task number.")
```

```
C:\Users\willi\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\willi\PycharmProjects\PythonProject\Homework.py
Welcome to the SPY Tracker!
We have prices for the last 5 days.
Enter the day number (1-5): 2
The SPY price on Day 2 was $588.75
Would you like to check another day? (y/n): y
Enter the day number (1-5): 3
The SPY price on Day 3 was $588.43
Would you like to check another day? (y/n): n
Thank you for using the SPY Price Tracker.

Process finished with exit code 0
```

```
C:\Users\willi\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\willi\PycharmProjects\PythonProject\Homework.py

To-Do List Menu:
1. View tasks
2. Add task
3. Mark task as completed
4. Remove task
5. Quit
Choose an option (1-5): 2
Enter the task: 23445433553
Task added.

To-Do List Menu:
1. View tasks
2. Add task
3. Mark task as completed
4. Remove task
5. Quit
Choose an option (1-5): 2
Enter the task: 342435465
Task added.

To-Do List Menu:
1. View tasks
2. Add task
3. Mark task as completed
4. Remove task
5. Quit
Choose an option (1-5): 3

Your to-do list:
1. [×] 23445433553
2. [×] 342435465
Enter the number of the task to mark as completed: 22
Invalid task number.

To-Do List Menu:
1. View tasks
2. Add task
```

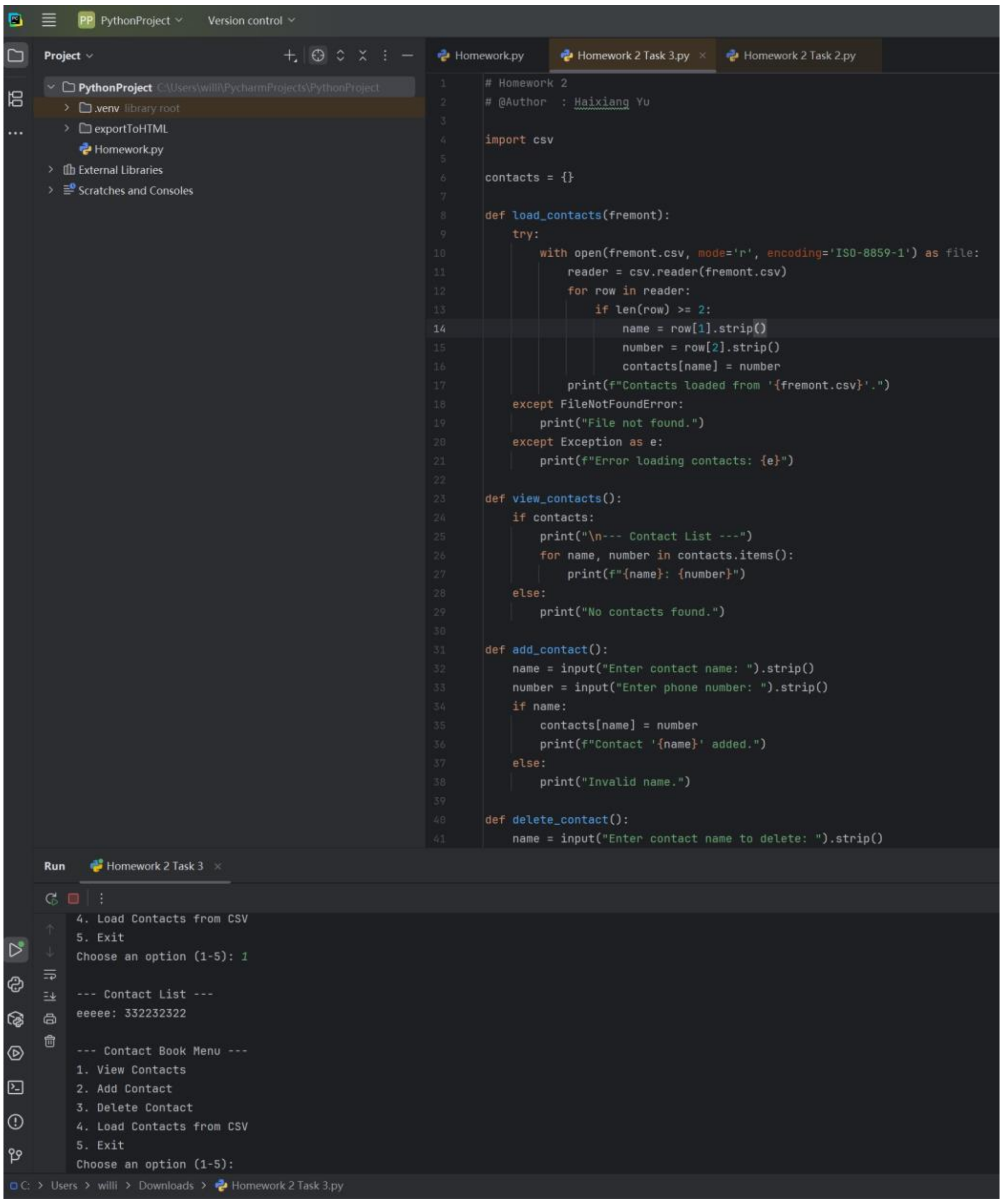
```
3. Mark task as completed
4. Remove task
5. Quit
Choose an option (1-5): 1

Your to-do list:
1. [×] 23445433553
2. [×] 342435465

To-Do List Menu:
1. View tasks
2. Add task
3. Mark task as completed
4. Remove task
5. Quit
Choose an option (1-5):
```



```python
# Homework 2
# @Author  : Haixiang Yu

# Tuple to store SPY stock prices for the past 5 days
SPY_prices = (572.41, 588.75, 588.43, 590.71, 594.26)

def show_price_for_day():  1 usage
    try:
        day = int(input("Enter the day number (1-5): "))
        if 1 <= day <= 5:
            price = SPY_prices[day - 1]
            print(f"The SPY price on Day {day} was ${price}")
        else:
            print("Invalid day. Please enter a number between 1 and 5.")
    except ValueError:
        print("Please enter a valid number.")

def main():  1 usage
    print("Welcome to the SPY Tracker!")
    print("We have prices for the last 5 days.")

    while True:
        show_price_for_day()
        again = input("Would you like to check another day? (y/n): ").lower()
        if again != 'y':
            print("Thank you for using the SPY Price Tracker.")
            break

if __name__ == "__main__":
    main()
```

```
C:\Users\willi\PycharmProjects\PythonProject\.venv\Scripts\python.exe C:\Users\willi\PycharmProjects\PythonProject\Homework.py
Welcome to the SPY Tracker!
We have prices for the last 5 days.
Enter the day number (1-5): 2
The SPY price on Day 2 was $588.75
Would you like to check another day? (y/n): y
Enter the day number (1-5): 3
The SPY price on Day 3 was $588.43
Would you like to check another day? (y/n): n
Thank you for using the SPY Price Tracker.

Process finished with exit code 0
```

Task 3

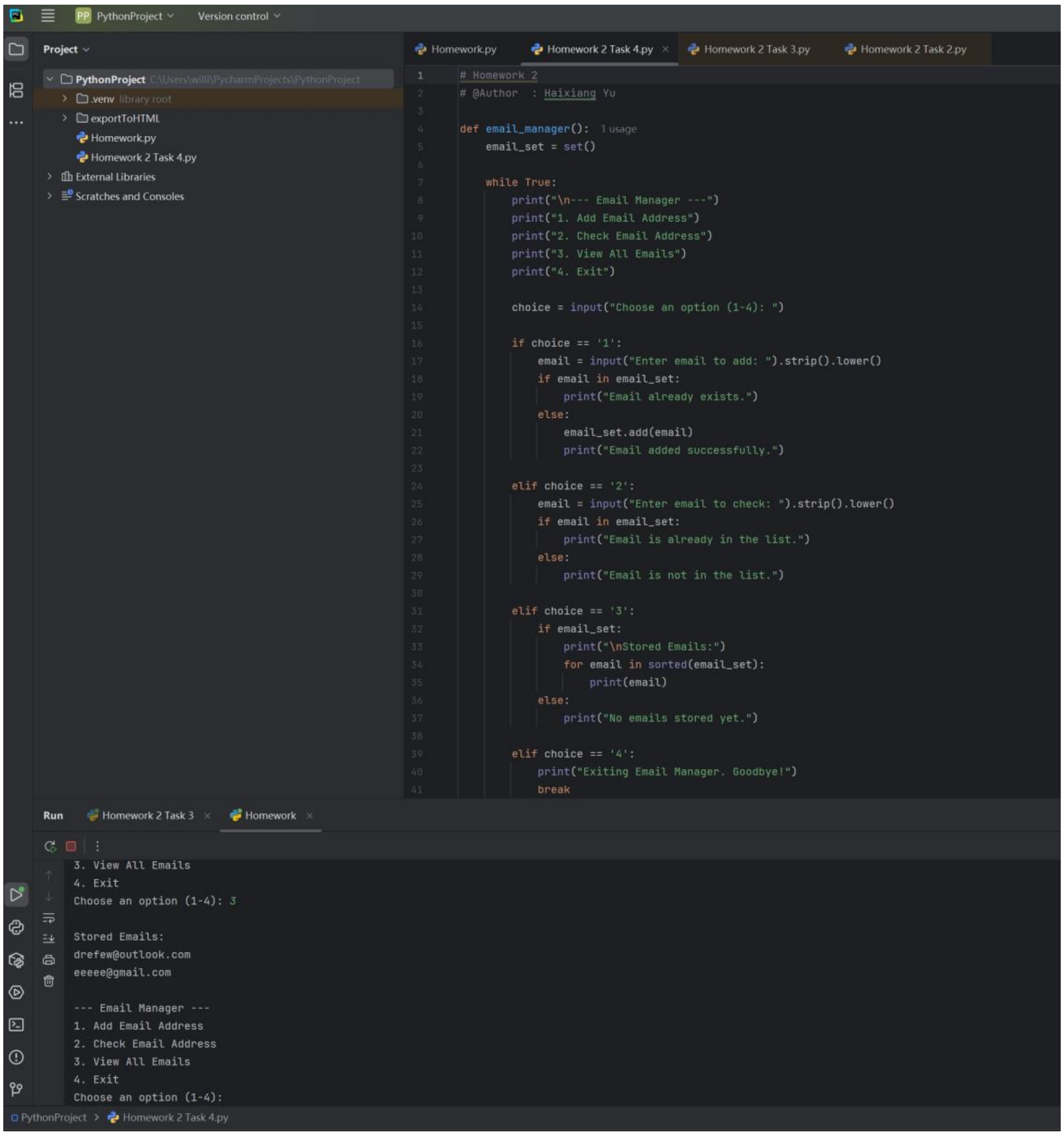


```python
# Homework 2
# @Author  : Haixiang Yu

import csv

contacts = {}

def load_contacts(fremont):
    try:
        with open(fremont.csv, mode='r', encoding='ISO-8859-1') as file:
            reader = csv.reader(fremont.csv)
            for row in reader:
                if len(row) >= 2:
                    name = row[1].strip()
                    number = row[2].strip()
                    contacts[name] = number
            print(f"Contacts loaded from '{fremont.csv}'.")
    except FileNotFoundError:
        print("File not found.")
    except Exception as e:
        print(f"Error loading contacts: {e}")

def view_contacts():
    if contacts:
        print("\n--- Contact List ---")
        for name, number in contacts.items():
            print(f"{name}: {number}")
    else:
        print("No contacts found.")

def add_contact():
    name = input("Enter contact name: ").strip()
    number = input("Enter phone number: ").strip()
    if name:
        contacts[name] = number
        print(f"Contact '{name}' added.")
    else:
        print("Invalid name.")

def delete_contact():
    name = input("Enter contact name to delete: ").strip()
```

```
4. Load Contacts from CSV
5. Exit
Choose an option (1-5): 1

--- Contact List ---
eeeee: 332232322

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
Choose an option (1-5):
```

```
C:\Users\willi\PycharmProjects\PythonProject\.venv\Scripts\python.exe "C:\Users\willi\Downloads\Homework 2 Task 3.py"

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
Choose an option (1-5): 1
No contacts found.

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
Choose an option (1-5): 2
Enter contact name: eeeee
Enter phone number: 332232322
Contact 'eeeee' added.

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
Choose an option (1-5): 4
Enter the CSV filename (e.g., fremont.csv): fremont
Error loading contacts: 'str' object has no attribute 'csv'

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
```

```
Choose an option (1-5): 1

--- Contact List ---
eeeee: 332232322

--- Contact Book Menu ---
1. View Contacts
2. Add Contact
3. Delete Contact
4. Load Contacts from CSV
5. Exit
Choose an option (1-5):
```



```python
# Homework 2
# @Author  : Haixiang Yu

def email_manager():  1 usage
    email_set = set()

    while True:
        print("\n--- Email Manager ---")
        print("1. Add Email Address")
        print("2. Check Email Address")
        print("3. View All Emails")
        print("4. Exit")

        choice = input("Choose an option (1-4): ")

        if choice == '1':
            email = input("Enter email to add: ").strip().lower()
            if email in email_set:
                print("Email already exists.")
            else:
                email_set.add(email)
                print("Email added successfully.")

        elif choice == '2':
            email = input("Enter email to check: ").strip().lower()
            if email in email_set:
                print("Email is already in the list.")
            else:
                print("Email is not in the list.")

        elif choice == '3':
            if email_set:
                print("\nStored Emails:")
                for email in sorted(email_set):
                    print(email)
            else:
                print("No emails stored yet.")

        elif choice == '4':
            print("Exiting Email Manager. Goodbye!")
            break
```

```
3. View All Emails
4. Exit
Choose an option (1-4): 3

Stored Emails:
drefew@outlook.com
eeeee@gmail.com

--- Email Manager ---
1. Add Email Address
2. Check Email Address
3. View All Emails
4. Exit
Choose an option (1-4):
```

```
C:\Users\willi\PycharmProjects\PythonProject\.venv\
Scripts\python.exe C:\Users\willi\PycharmProjects\
PythonProject\Homework.py

--- Email Manager ---
1. Add Email Address
2. Check Email Address
3. View All Emails
4. Exit
Choose an option (1-4): 1
Enter email to add: eeeee@gmail.com
Email added successfully.

--- Email Manager ---
1. Add Email Address
2. Check Email Address
3. View All Emails
4. Exit
Choose an option (1-4): 1
Enter email to add: drefew@outlook.com
Email added successfully.

--- Email Manager ---
1. Add Email Address
2. Check Email Address
3. View All Emails
4. Exit
Choose an option (1-4): 3

Stored Emails:
drefew@outlook.com
eeeee@gmail.com

--- Email Manager ---
1. Add Email Address
2. Check Email Address
3. View All Emails
4. Exit
Choose an option (1-4):
```

Haixiang Yu

PRG 501

Homework 2 Part 2

Real-World Applications of Data Structures: lists, dictionaries and collections

Data structures are fundamental components of software development that enable efficient organization, retrieval, and manipulation of data. This report explores three widely used data structures - lists, dictionaries, and collections - and analyzes their practical applications, strengths, and limitations. By analyzing real-world examples from industries such as social media, e-commerce, and web development, we gain insight into why particular data structures are chosen for specific tasks.

Lists have a large number of applications in reality. Lists are ordered collections of items that allow repetition and are indexed by location. They are commonly used in social media applications to store friends, followers, and posts. For example, Instagram might use lists to maintain a user's friends list, while Tiktok uses lists to manage post dynamics sorted by time or relevance. In e-commerce platforms like Amazon, shopping carts are often implemented using lists, where each item added by a user is added to the list.

Lists are highly flexible and easy to maintain ordered collections. Their ability to store duplicate items and maintain insertion order makes them ideal for managing user-specific data such as timelines and shopping carts. Lists are mutable, which means that their contents can be modified, allowing for dynamic updates as users add or remove items. A major limitation of lists is the performance degradation when dealing with large datasets, especially when frequent search or delete operations need to be performed. Since list elements are accessed through an index, searching for a specific item requires a linear search, resulting in a time complexity of O(n). This can lead to inefficiencies in large-scale applications unless optimized using other data structures.

Dictionaries also have examples of applications in practice. Dictionaries (or key-value pairs) are widely used in Web development to manage user profiles and session data. Each user can be identified by a unique key (e.g., user ID), and their associated data (name, preferences, settings) is stored as values. In content management systems such as WordPress, dictionaries are used to store configuration settings and metadata for pages and posts. Dictionaries provide constant-time (O(1)) access to data using keys, making them very efficient to find and update. This is especially useful in web applications that require fast retrieval of user data. The structure of the dictionary supports a clear and logical representation of data, enabling developers to efficiently model real-world entities. In versions of Python prior to 3.7, dictionaries are unorganized, which can be a limitation when data order is important. Dictionaries also consume more memory than lists due to the overhead of storing keys. In addition, keys must be unique and immutable, which may need to be handled carefully in dynamic applications.

A collection is an unordered and unique set of items, often used in applications where duplication needs to be eliminated. For example, email marketing platforms use collections to maintain mailing lists to ensure that no recipient receives the same email multiple times. In cybersecurity, collections are used to store blacklisted IP addresses or hashes of malicious files to enable fast membership checks. The main advantage of ensembles is their enforced uniqueness and efficient membership testing, usually within O(1) time complexity. This makes them well suited for operations such as removing duplicate items, checking for common elements between two datasets, and enforcing data integrity. Collections cannot maintain order or store duplicate elements, which can be a disadvantage in scenarios where data order or duplication is critical. They are also limited to storing immutable elements, which limits their flexibility in some applications.

Conclusion, understanding the advantages and limitations of different data structures is essential for designing efficient software systems. Lists provide flexibility and organized data storage, dictionaries support fast lookups and structured data representation, and collections ensure uniqueness and support high-performance membership testing. By appropriately applying these structures, developers can create applications that are not only fully functional but also optimized for performance and scalability.