Data Structures and Algorithms (DSA)

Lab Report

Name:	Iqra Fatima
Reg. Number:	23-CP-62
Semester:	4th
Department:	CPED

Submitted To:

Engineer Sheharyar Khan

Lab Report

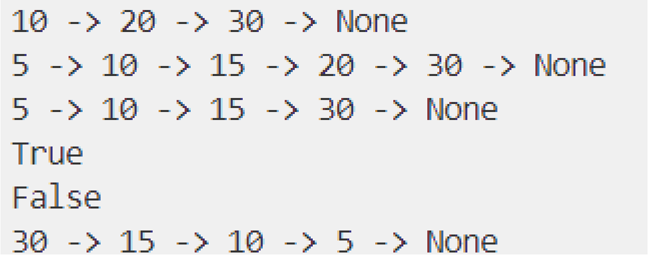
Examples

Task 1-5:

Code:

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class SinglyLinkedList:
    def __init__(self):
        self.head = None

    def append(self, data):
        node = Node(data)
        if not self.head:
            self.head = node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = node

    def insert_at_beginning(self, data):
        node = Node(data)
        node.next = self.head
        self.head = node

    def insert_at_position(self, data, pos):
        if pos == 0:
            self.insert_at_beginning(data)
            return
        node = Node(data)
        current = self.head
        for _ in range(pos - 1):
            if not current.next:
                break
            current = current.next
        node.next = current.next
        current.next = node

    def delete(self, data):
        current = self.head
        prev = None
        while current:
            if current.data == data:
                if prev:
                    prev.next = current.next
                else:
                    self.head = current.next
                return
            prev = current
            current = current.next

    def search(self, data):
        current = self.head
        while current:
            if current.data == data:
                return True
            current = current.next
        return False

    def reverse(self):
        prev = None
        current = self.head
        while current:
            next_node = current.next
            current.next = prev
            prev = current
            current = next_node
        self.head = prev

    def display(self):
        current = self.head
        while current:
            print(current.data, end=" -> ")
            current = current.next
        print("None")

# Example Usage
sll = SinglyLinkedList()
sll.append(10)
sll.append(20)
sll.append(30)
sll.display()

sll.insert_at_beginning(5)
sll.insert_at_position(15, 2)
sll.display()

sll.delete(20)
sll.display()

print(sll.search(15))
print(sll.search(100))

sll.reverse()
sll.display()
```

Output:



```
10 -> 20 -> 30 -> None
5 -> 10 -> 15 -> 20 -> 30 -> None
5 -> 10 -> 15 -> 30 -> None
True
False
30 -> 15 -> 10 -> 5 -> None
```

Problems

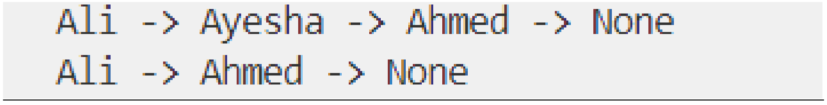
Easy Problems

1. Student Name List: Store and manage a list of student names in a linked list.

Code:

```
class Node:
    def __init__(self, name):
        self.name = name
        self.next = None

class StudentList:
    def __init__(self):
        self.head = None

    def add_student(self, name):
        new_node = Node(name)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node

    def remove_student(self, name):
        if not self.head:
            print("List is empty.")
            return

        if self.head.name == name:
            self.head = self.head.next
            return

        temp = self.head
        while temp.next and temp.next.name != name:
            temp = temp.next
        if temp.next:
            temp.next = temp.next.next
        else:
            print("Student not found.")

    def display_students(self):
        temp = self.head
        while temp:
            print(temp.name, end=" -> ")
            temp = temp.next
        print("None")

# Example Usage
students = StudentList()
students.add_student("Ali")
students.add_student("Ayesha")
students.add_student("Ahmed")
students.display_students()

students.remove_student("Ayesha")
students.display_students()
```

Output:



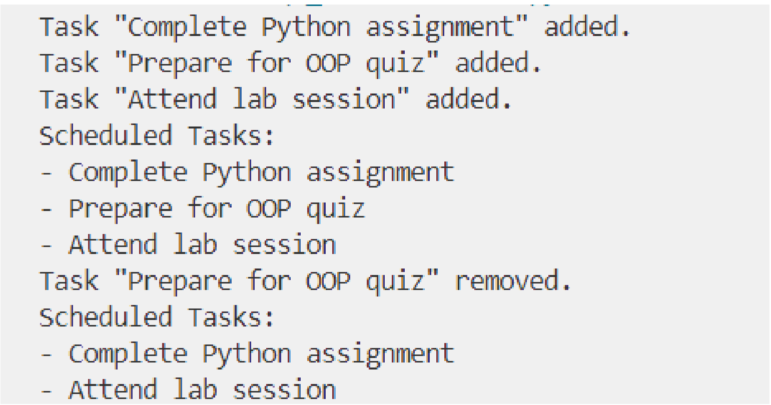
```
Ali -> Ayesha -> Ahmed -> None
Ali -> Ahmed -> None
```

2. Task Scheduler: Implement a simple task manager where users can add/remove tasks.

Code:

```
class TaskNode:
    def __init__(self, task):
        self.task = task
        self.next = None

class TaskScheduler:
    def __init__(self):
        self.head = None

    def add_task(self, task):
        new_node = TaskNode(task)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print(f'Task "{task}" added.')
    def remove_task(self, task):
        if not self.head:
            print("Task list is empty.")
            return

        if self.head.task == task:
            self.head = self.head.next
            print(f'Task "{task}" removed.')
            return

        temp = self.head
        while temp.next and temp.next.task != task:
            temp = temp.next

        if temp.next:
            temp.next = temp.next.next
            print(f'Task "{task}" removed.')
        else:
            print(f'Task "{task}" not found.')

    def display_tasks(self):
        if not self.head:
            print("No tasks scheduled.")
            return

        print("Scheduled Tasks:")
        temp = self.head
        while temp:
            print(f"- {temp.task}")
            temp = temp.next

# Example Usage
tasks = TaskScheduler()
tasks.add_task("Complete Python assignment")
tasks.add_task("Prepare for OOP quiz")
tasks.add_task("Attend lab session")
tasks.display_tasks()
tasks.remove_task("Prepare for OOP quiz")
tasks.display_tasks()
```

Output:



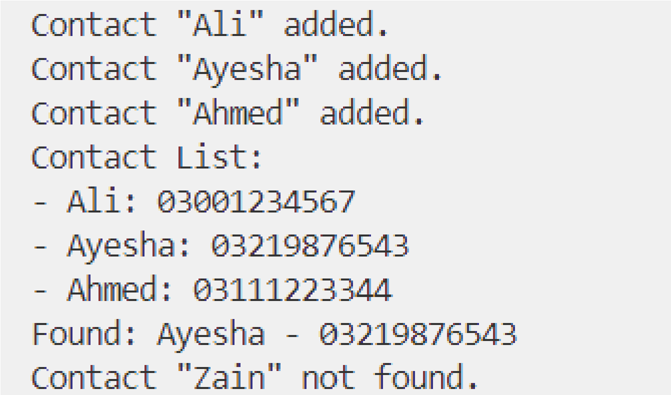
```
Task "Complete Python assignment" added.
Task "Prepare for OOP quiz" added.
Task "Attend lab session" added.
Scheduled Tasks:
- Complete Python assignment
- Prepare for OOP quiz
- Attend lab session
Task "Prepare for OOP quiz" removed.
Scheduled Tasks:
- Complete Python assignment
- Attend lab session
```

3. Contact List: Create a contact list using linked lists where users can search by name.

Code:

```
class ContactNode:
    def __init__(self, name, phone):
        self.name = name
        self.phone = phone
        self.next = None

class ContactList:
    def __init__(self):
        self.head = None

    def add_contact(self, name, phone):
        new_node = ContactNode(name, phone)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print(f'Contact "{name}" added.')

    def search_contact(self, name):
        temp = self.head
        while temp:
            if temp.name.lower() == name.lower():
                print(f'Found: {temp.name} - {temp.phone}')
                return
            temp = temp.next
        print(f'Contact "{name}" not found.')

    def display_contacts(self):
        if not self.head:
            print("No contacts available.")
            return

        print("Contact List:")
        temp = self.head
        while temp:
            print(f"- {temp.name}: {temp.phone}")
            temp = temp.next

# Example Usage
contacts = ContactList()
contacts.add_contact("Ali", "03001234567")
contacts.add_contact("Ayesha", "03219876543")
contacts.add_contact("Ahmed", "03111223344")
contacts.display_contacts()
contacts.search_contact("Ayesha")
contacts.search_contact("Zain")
```

<u>Output:</u>



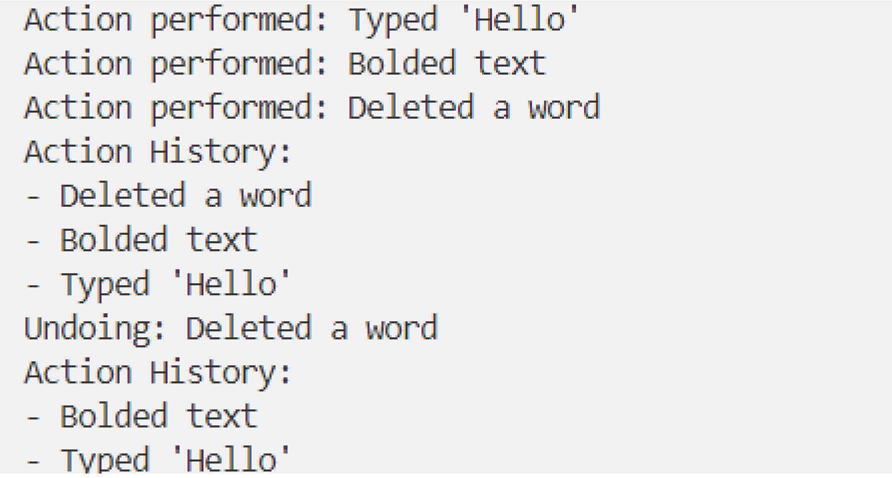
```
Contact "Ali" added.
Contact "Ayesha" added.
Contact "Ahmed" added.
Contact List:
- Ali: 03001234567
- Ayesha: 03219876543
- Ahmed: 03111223344
Found: Ayesha - 03219876543
Contact "Zain" not found.
```

4. Undo Feature in Editor: Implement a basic undo feature where previous actions are stored.

<u>Code:</u>

```
class ActionNode:
    def __init__(self, action):
        self.action = action
        self.next = None

class UndoFeature:
    def __init__(self):
        self.top = None  # Stack کے لیے ٹاپ پوائنٹر

    def perform_action(self, action):
        new_node = ActionNode(action)
        new_node.next = self.top
        self.top = new_node
        print(f'Action performed: {action}')

    def undo(self):
        if not self.top:
            print("No actions to undo.")
            return

        print(f'Undoing: {self.top.action}')
        self.top = self.top.next  # پچھلے ایکشن پر واپس جانا

    def display_actions(self):
        if not self.top:
            print("No actions recorded.")
            return

        print("Action History:")
        temp = self.top
        while temp:
            print(f"- {temp.action}")
            temp = temp.next

# Example Usage
editor = UndoFeature()
editor.perform_action("Typed 'Hello'")
editor.perform_action("Bolded text")
editor.perform_action("Deleted a word")

editor.display_actions()

editor.undo()
editor.display_actions()
```

Output:



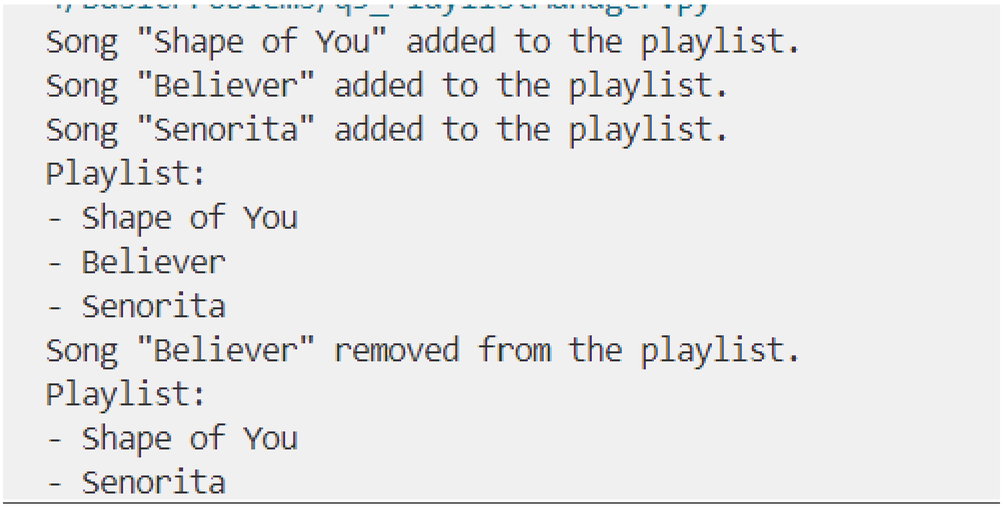
```
Action performed: Typed 'Hello'
Action performed: Bolded text
Action performed: Deleted a word
Action History:
- Deleted a word
- Bolded text
- Typed 'Hello'
Undoing: Deleted a word
Action History:
- Bolded text
- Typed 'Hello'
```

5. Simple Playlist Manager: Store a list of songs and provide a method to display them.

Code:

```
class SongNode:
    def __init__(self, title):
        self.title = title
        self.next = None

class Playlist:
    def __init__(self):
        self.head = None

    def add_song(self, title):
        new_node = SongNode(title)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print(f'Song "{title}" added to the playlist.')

    def remove_song(self, title):
        if not self.head:
            print("Playlist is empty.")
            return

        if self.head.title == title:
            self.head = self.head.next
            print(f'Song "{title}" removed from the playlist.')
            return

        temp = self.head
        while temp.next and temp.next.title != title:
            temp = temp.next

        if temp.next:
            temp.next = temp.next.next
            print(f'Song "{title}" removed from the playlist.')
        else:
            print(f'Song "{title}" not found in the playlist.')

    def display_playlist(self):
        if not self.head:
            print("No songs in the playlist.")
            return

        print("Playlist:")
        temp = self.head
        while temp:
            print(f"- {temp.title}")
            temp = temp.next

# Example Usage
playlist = Playlist()
playlist.add_song("Shape of You")
playlist.add_song("Believer")
playlist.add_song("Senorita")

playlist.display_playlist()

playlist.remove_song("Believer")
playlist.display_playlist()
```

Output:



```
Song "Shape of You" added to the playlist.
Song "Believer" added to the playlist.
Song "Senorita" added to the playlist.
Playlist:
- Shape of You
- Believer
- Senorita
Song "Believer" removed from the playlist.
Playlist:
- Shape of You
- Senorita
```

Intermediate Problems

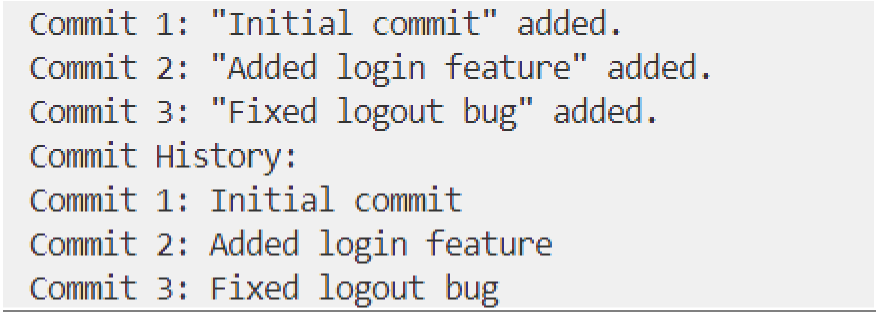
1. Version Control System: Simulate a Git commit history where commits are stored

in a linked list.

Code:

```
class CommitNode:
    def __init__(self, commit_id, message):
        self.commit_id = commit_id
        self.message = message
        self.next = None

class VersionControlSystem:
    def __init__(self):
        self.head = None
        self.commit_count = 0

    def add_commit(self, message):
        self.commit_count += 1
        new_node = CommitNode(self.commit_count, message)
        if not self.head:
            self.head = new_node
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_node
        print(f'Commit {self.commit_count}: "{message}" added.')

    def display_commits(self):
        if not self.head:
            print("No commits found.")
            return

        print("Commit History:")
        temp = self.head
        while temp:
            print(f'Commit {temp.commit_id}: {temp.message}')
            temp = temp.next

# Example Usage
vcs = VersionControlSystem()
vcs.add_commit("Initial commit")
vcs.add_commit("Added login feature")
vcs.add_commit("Fixed logout bug")

vcs.display_commits()
```

Output:



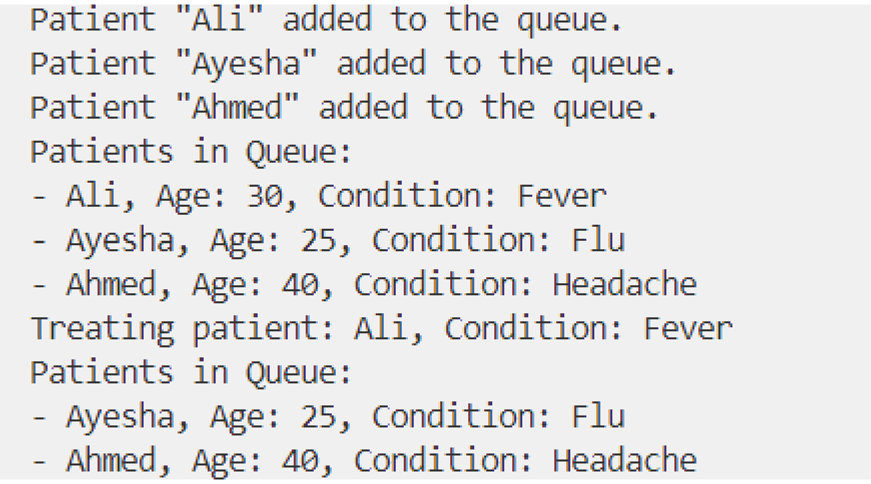
```
Commit 1: "Initial commit" added.
Commit 2: "Added login feature" added.
Commit 3: "Fixed logout bug" added.
Commit History:
Commit 1: Initial commit
Commit 2: Added login feature
Commit 3: Fixed logout bug
```

2. Hospital Patient Queue: Implement a queue system where patients are treated in order.

Code:

```
class PatientNode:
    def __init__(self, name, age, condition):
        self.name = name
        self.age = age
        self.condition = condition
        self.next = None

class HospitalQueue:
    def __init__(self):
        self.front = None  # Queue کا پہلا مریض
        self.rear = None   # Queue کا آخری مریض

    def add_patient(self, name, age, condition):
        new_node = PatientNode(name, age, condition)
        if not self.rear:  # اگر queue خالی ہے
            self.front = self.rear = new_node
        else:
            self.rear.next = new_node
            self.rear = new_node
        print(f'Patient "{name}" added to the queue.')

    def treat_patient(self):
        if not self.front:
            print("No patients in the queue.")
            return

        print(f'Treating patient: {self.front.name}, Condition: 
{self.front.condition}')
        self.front = self.front.next  # اگلا مریض queue میں آئے گا

        if not self.front:  # اگر queue خالی ہو جائے
            self.rear = None

    def display_patients(self):
        if not self.front:
            print("No patients in the queue.")
            return

        print("Patients in Queue:")
        temp = self.front
        while temp:
            print(f'- {temp.name}, Age: {temp.age}, Condition: 
{temp.condition}')
            temp = temp.next

# Example Usage
hospital = HospitalQueue()
hospital.add_patient("Ali", 30, "Fever")
hospital.add_patient("Ayesha", 25, "Flu")
hospital.add_patient("Ahmed", 40, "Headache")

hospital.display_patients()
hospital.treat_patient()
hospital.display_patients()
```

Output:



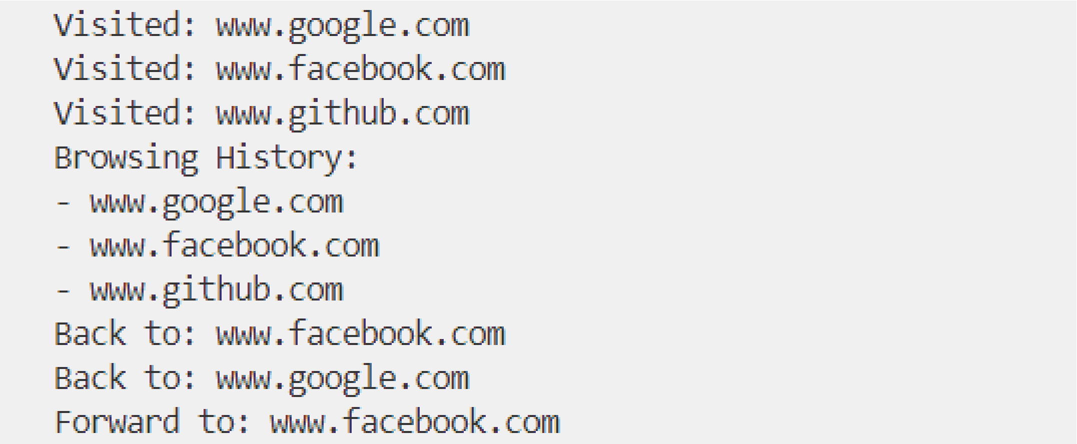
```
Patient "Ali" added to the queue.
Patient "Ayesha" added to the queue.
Patient "Ahmed" added to the queue.
Patients in Queue:
- Ali, Age: 30, Condition: Fever
- Ayesha, Age: 25, Condition: Flu
- Ahmed, Age: 40, Condition: Headache
Treating patient: Ali, Condition: Fever
Patients in Queue:
- Ayesha, Age: 25, Condition: Flu
- Ahmed, Age: 40, Condition: Headache
```

3. Web Browser Navigation: Implement a forward/backward navigation in a web browser.

Code:

```
class PageNode:
    def __init__(self, url):
        self.url = url
        self.next = None
        self.prev = None

class BrowserNavigation:
    def __init__(self):
        self.current_page = None
        self.history = None

    def visit_page(self, url):
        new_page = PageNode(url)
        if not self.current_page:
            self.history = self.current_page = new_page
        else:
            self.current_page.next = new_page
            new_page.prev = self.current_page
            self.current_page = new_page
        print(f'Visited: {url}')

    def go_back(self):
        if not self.current_page or not self.current_page.prev:
            print("No previous page.")
            return
        self.current_page = self.current_page.prev
        print(f'Back to: {self.current_page.url}')

    def go_forward(self):
        if not self.current_page or not self.current_page.next:
            print("No forward page.")
            return
        self.current_page = self.current_page.next
        print(f'Forward to: {self.current_page.url}')

    def display_history(self):
        if not self.history:
            print("No browsing history.")
            return

        print("Browsing History:")
        temp = self.history
        while temp:
            print(f'- {temp.url}')
            temp = temp.next
# Example Usage
browser = BrowserNavigation()
browser.visit_page("www.google.com")
browser.visit_page("www.facebook.com")
browser.visit_page("www.github.com")
browser.display_history()
browser.go_back()
browser.go_back()
browser.go_forward()
```

Output:



```
Visited: www.google.com
Visited: www.facebook.com
Visited: www.github.com
Browsing History:
- www.google.com
- www.facebook.com
- www.github.com
Back to: www.facebook.com
Back to: www.google.com
Forward to: www.facebook.com
```

4. File Management System: Simulate a hierarchical file system using linked lists.

Code:

```
class FileNode:
    def __init__(self, file_name):
        self.file_name = file_name
        self.next = None

class Directory:
    def __init__(self, directory_name):
        self.directory_name = directory_name
        self.files = None

    def create_file(self, file_name):
        new_file = FileNode(file_name)
        if not self.files:
            self.files = new_file
        else:
            temp = self.files
            while temp.next:
                temp = temp.next
            temp.next = new_file
        print(f'File "{file_name}" created in
{self.directory_name}.')

    def delete_file(self, file_name):
        if not self.files:
            print("No files in this directory.")
            return

        if self.files.file_name == file_name:
            self.files = self.files.next
            print(f'File "{file_name}" deleted from 
{self.directory_name}.')
            return

        temp = self.files
        while temp.next and temp.next.file_name != file_name:
            temp = temp.next

        if temp.next:
            temp.next = temp.next.next
            print(f'File "{file_name}" deleted from 
{self.directory_name}.')
        else:
            print(f'File "{file_name}" not found in 
{self.directory_name}.')

    def display_files(self):
        if not self.files:
            print(f"No files in {self.directory_name}.")
            return

        print(f"Files in {self.directory_name}:")
        temp = self.files
        while temp:
            print(f'- {temp.file_name}')
            temp = temp.next

# Example Usage
directory = Directory("Documents")
directory.create_file("file1.txt")
directory.create_file("file2.txt")
directory.create_file("file3.txt")

directory.display_files()

directory.delete_file("file2.txt")
directory.display_files()
```

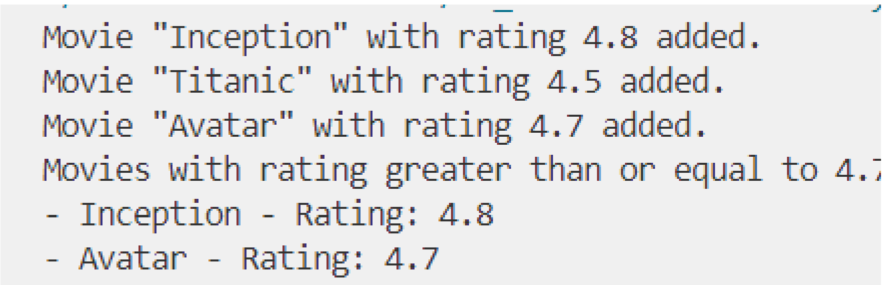
Output:

5. Movie Recommendation System: Store user ratings and suggest similar movies.

Code:

```
class MovieNode:
    def __init__(self, movie_name, rating):
        self.movie_name = movie_name
        self.rating = rating
        self.next = None

class MovieRecommendationSystem:
    def __init__(self):
        self.head = None

    def add_movie(self, movie_name, rating):
        new_movie = MovieNode(movie_name, rating)
        if not self.head:
            self.head = new_movie
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_movie
        print(f'Movie "{movie_name}" with rating {rating} added.')

    def recommend_movies(self, min_rating):
        temp = self.head
        found = False
        print(f"Movies with rating greater than or equal to
{min_rating}:")
        while temp:
            if temp.rating >= min_rating:
                print(f'- {temp.movie_name} - Rating:
{temp.rating}')
                found = True
            temp = temp.next
        if not found:
            print("No movies found with the specified rating.")

# Example Usage
movie_system = MovieRecommendationSystem()
movie_system.add_movie("Inception", 4.8)
movie_system.add_movie("Titanic", 4.5)
movie_system.add_movie("Avatar", 4.7)

movie_system.recommend_movies(4.7)
```

Output:



```
Movie "Inception" with rating 4.8 added.
Movie "Titanic" with rating 4.5 added.
Movie "Avatar" with rating 4.7 added.
Movies with rating greater than or equal to 4.7
- Inception - Rating: 4.8
- Avatar - Rating: 4.7
```

Advanced Problems

1. Facebook Messenger Chat History

o Implement a chat system where messages are stored in a linked list and

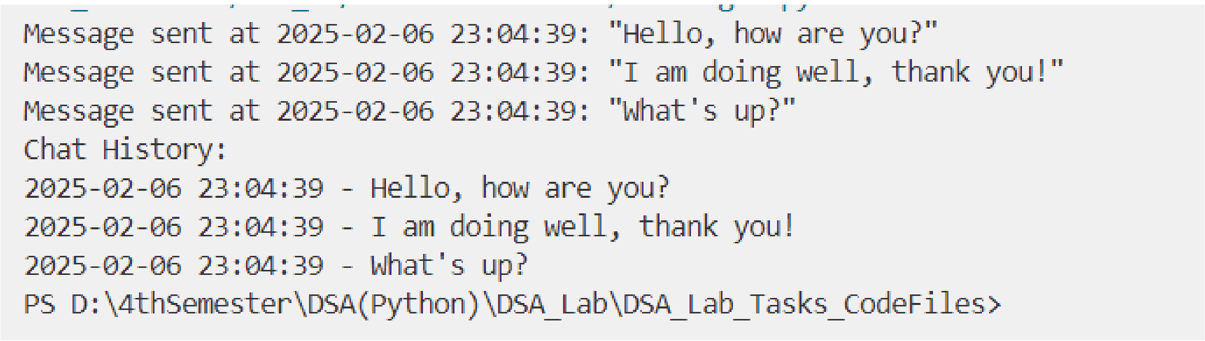
retrieved in order.

o Hint: Store messages as nodes with timestamps.

Code:

```
from datetime import datetime

class MessageNode:
    def __init__(self, message, timestamp):
        self.message = message
        self.timestamp = timestamp
        self.next = None

class ChatHistory:
    def __init__(self):
        self.head = None

    def send_message(self, message):
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        new_message = MessageNode(message, timestamp)
        if not self.head:
            self.head = new_message
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_message
        print(f'Message sent at {timestamp}: "{message}"')

    def display_chat(self):
        if not self.head:
            print("No chat history.")
            return

        print("Chat History:")
        temp = self.head
        while temp:
            print(f'{temp.timestamp} - {temp.message}')
            temp = temp.next

# Example Usage
chat = ChatHistory()
chat.send_message("Hello, how are you?")
chat.send_message("I am doing well, thank you!")
chat.send_message("What's up?")
chat.display_chat()
```

Output:



```
Message sent at 2025-02-06 23:04:39: "Hello, how are you?"
Message sent at 2025-02-06 23:04:39: "I am doing well, thank you!"
Message sent at 2025-02-06 23:04:39: "What's up?"
Chat History:
2025-02-06 23:04:39 - Hello, how are you?
2025-02-06 23:04:39 - I am doing well, thank you!
2025-02-06 23:04:39 - What's up?
PS D:\4thSemester\DSA(Python)\DSA_Lab\DSA_Lab_Tasks_CodeFiles>
```

2. LinkedIn Profile Connections

o Implement a user profile system where each user is a node connected to other

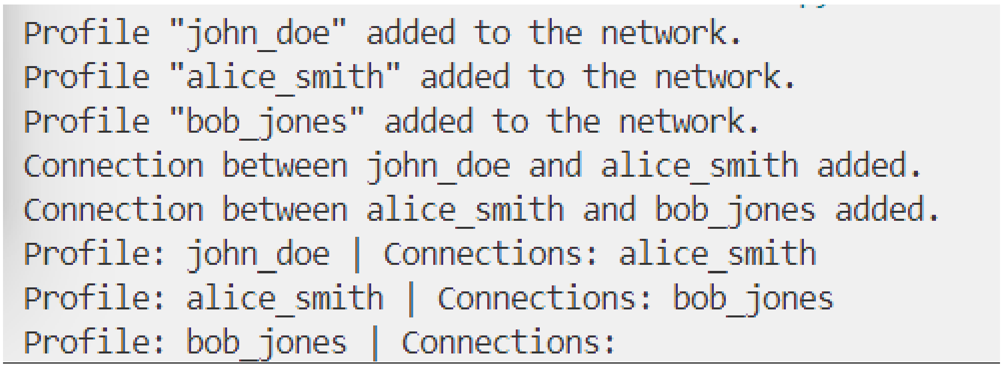
users.

o Hint: Each node contains a list of connections.

Code:

```
class ProfileNode:
    def __init__(self, username):
        self.username = username
        self.connections = []  # لیسٹ میں کنکشنز رکھے جائیں گے
        self.next = None

class LinkedInNetwork:
    def __init__(self):
        self.head = None

    def add_profile(self, username):
        new_profile = ProfileNode(username)
        if not self.head:
            self.head = new_profile
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_profile
        print(f'Profile "{username}" added to the network.')

    def add_connection(self, username, connection_username):
        temp = self.head
        while temp:
            if temp.username == username:
                temp.connections.append(connection_username)
                print(f'Connection between {username} and
{connection_username} added.')
                return
            temp = temp.next
        print(f'Profile {username} not found.')
    def display_network(self):
        if not self.head:
            print("No profiles in the network.")
            return

        temp = self.head
        while temp:
            print(f'Profile: {temp.username} | Connections: {",
".join(temp.connections)}')
            temp = temp.next

# Example Usage
network = LinkedInNetwork()
network.add_profile("john_doe")
network.add_profile("alice_smith")
network.add_profile("bob_jones")

network.add_connection("john_doe", "alice_smith")
network.add_connection("alice_smith", "bob_jones")

network.display_network()
```

Output:



```
Profile "john_doe" added to the network.
Profile "alice_smith" added to the network.
Profile "bob_jones" added to the network.
Connection between john_doe and alice_smith added.
Connection between alice_smith and bob_jones added.
Profile: john_doe | Connections: alice_smith
Profile: alice_smith | Connections: bob_jones
Profile: bob_jones | Connections:
```

3. Google Docs Edit History

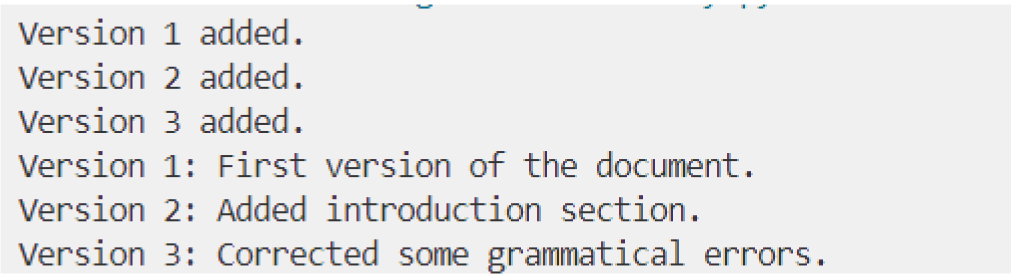
o Simulate edit history tracking in Google Docs.

o Hint: Each node stores a version of the document.

Code:

```
class DocumentNode:
    def __init__(self, version, content):
        self.version = version
        self.content = content
        self.next = None

class GoogleDocsHistory:
    def __init__(self):
        self.head = None


    def add_version(self, content):
        version = 1
        if self.head:
            temp = self.head
            while temp.next:
                temp = temp.next
            version = temp.version + 1
        new_version = DocumentNode(version, content)
        if not self.head:
            self.head = new_version
        else:
            temp.next = new_version
        print(f'Version {version} added.')

    def display_history(self):
        if not self.head:
            print("No version history available.")
            return

        temp = self.head
        while temp:
            print(f'Version {temp.version}: {temp.content}')
            temp = temp.next

# Example Usage
docs = GoogleDocsHistory()
docs.add_version("First version of the document.")
docs.add_version("Added introduction section.")
docs.add_version("Corrected some grammatical errors.")
docs.display_history()
```

Output:



```
Version 1 added.
Version 2 added.
Version 3 added.
Version 1: First version of the document.
Version 2: Added introduction section.
Version 3: Corrected some grammatical errors.
```

4. Pathfinding Algorithm in Maps

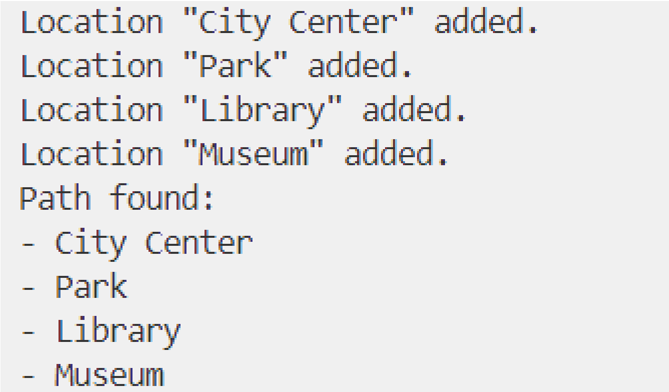
o Store a series of locations in a linked list and allow traversal.

o Hint: Each node represents a location.

Code:

```
class LocationNode:
    def __init__(self, location_name):
        self.location_name = location_name
        self.next = None

class Pathfinding:
    def __init__(self):
        self.head = None

    def add_location(self, location_name):
        new_location = LocationNode(location_name)
        if not self.head:
            self.head = new_location
        else:
            temp = self.head
            while temp.next:
                temp = temp.next
            temp.next = new_location
        print(f'Location "{location_name}" added.')

    def find_path(self, start_location, end_location):
        temp = self.head
        path = []
        while temp:
            path.append(temp.location_name)
            if temp.location_name == end_location:
                break
            temp = temp.next

        if end_location not in path:
            print("Path not found.")
            return

        print("Path found:")
        for loc in path:
            print(f'- {loc}')

# Example Usage
path = Pathfinding()
path.add_location("City Center")
path.add_location("Park")
path.add_location("Library")
path.add_location("Museum")

path.find_path("Park", "Museum")
```

Output:



```
Location "City Center" added.
Location "Park" added.
Location "Library" added.
Location "Museum" added.
Path found:
- City Center
- Park
- Library
- Museum
```

5. Blockchain Implementation

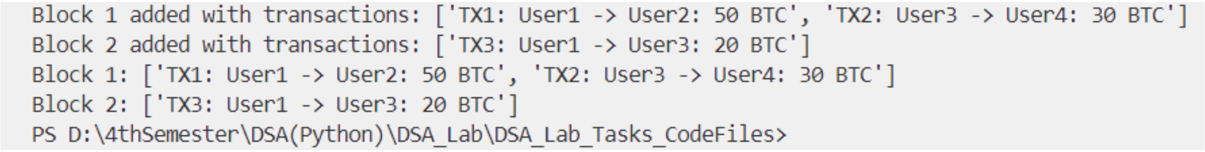
o Simulate a simple blockchain where each block stores transactions.

o Hint: Use linked list nodes to represent blocks.

Code:

```
class BlockNode:
    def __init__(self, block_number, transactions):
        self.block_number = block_number
        self.transactions = transactions
        self.next = None
class Blockchain:
    def __init__(self):
        self.head = None

    def add_block(self, transactions):
        block_number = 1
        if self.head:
            temp = self.head
            while temp.next:
                temp = temp.next
            block_number = temp.block_number + 1
        new_block = BlockNode(block_number, transactions)
        if not self.head:
            self.head = new_block
        else:
            temp.next = new_block

        print(f'Block {block_number} added with transactions: 
{transactions}')
    def display_chain(self):
        if not self.head:
            print("No blocks in the chain.")
            return
        temp = self.head
        while temp:
            print(f'Block {temp.block_number}: {temp.transactions}')
            temp = temp.next
# Example Usage
blockchain = Blockchain()
blockchain.add_block(["TX1: User1 -> User2: 50 BTC", "TX2: User3 -> 
User4: 30 BTC"])
blockchain.add_block(["TX3: User1 -> User3: 20 BTC"])

blockchain.display_chain()
```

Output:



```
Block 1 added with transactions: ['TX1: User1 -> User2: 50 BTC', 'TX2: User3 -> User4: 30 BTC']
Block 2 added with transactions: ['TX3: User1 -> User3: 20 BTC']
Block 1: ['TX1: User1 -> User2: 50 BTC', 'TX2: User3 -> User4: 30 BTC']
Block 2: ['TX3: User1 -> User3: 20 BTC']
PS D:\4thSemester\DSA(Python)\DSA_Lab\DSA_Lab_Tasks_CodeFiles>
```