Weekly Lab

Review

In this lab, we will review fundamental techniques in Python programming, including working with lists, strings, linked lists, stacks, and queues.

Exercise 1: Lists

Write a Python program with the following requirements:

1. **Input List**: Write a function to input a list from the keyboard.

```
def input_list(n: int) -> list:
```

2. **Print List**: Write a function to print the elements of the list.

```
def print_list(arr: list) -> None:
```

3. **Find Maximum Value**: Write a function to find the maximum value in the list.

```
def find_max(arr: list) -> int:
```

4. **Sum of List**: Write a function to calculate the sum of the elements in the list.

```
def sum_list(arr: list) -> int:
```

5. **Concatenate Lists**: Write a function to concatenate two lists into a new list.

```
def concat_lists(a: list, b: list) -> list:
```

6. **Longest Ascending Sublist**: Write a function to find the longest ascending sublist.

```
def find_longest_ascending_sublist(arr: list) -> list:
```

7. **Main Function**: In the `main` function, perform the following tasks:
   - Enter the number of elements in list `a` and input the list.
   - Enter the number of elements in list `b` and input the list.
   - Print the concatenated list `c`.
   - Print the maximum value in list `c`.
   - Print the sum of elements in list `c`.
   - Print the longest ascending sublist in list `c`.

Exercise 2: 2D Lists (Matrix)

Write a Python program with the following requirements:

1. **Read Matrix from File**: Write a function to read a matrix from a file.

   ```python
   def read_matrix(filename: str) -> list:
   ```

   Each row of the matrix should be on a separate line, with elements separated by spaces.

   For example:

   ```
   2 3
   1 2 3
   4 5 6
   ```

2. **Print Matrix to File**: Write a function to print a matrix to a file.

   Each row of the matrix should be on a separate line, with elements separated by spaces.

   For example:

   ```
   2 3
   1 2 3
   4 5 6
   ```

   ```python
   def print_matrix(filename: str, matrix: list) -> None:
   ```

3. **Matrix Multiplication**: Write a function to multiply two matrices.

   ```python
   def multiply_matrices(a: list, b: list) -> list:
   ```

4. **Matrix Transposition**: Write a function to compute the transpose of a matrix.

   ```python
   def transpose_matrix(matrix: list) -> list:
   ```

5. **Main Function**: In the `main` function, perform the following tasks:

   - Read matrix `a` from `matrix_a.txt`.
   - Read matrix `b` from `matrix_b.txt`.
   - Multiply matrices `a` and `b` and check if multiplication is valid.
   - Save the resulting matrix `c` to `matrix_c.txt`.
   - Compute the transpose of matrix `c` and save it to `matrix_c_transposed.txt`.

Exercise 3: String Processing in Python

Write a Python program to process a given paragraph of text and perform the following tasks:

Tasks:

1. Normalize the input paragraph by:
   - Removing extra spaces (leading, trailing, and multiple spaces between words).
   - Ensuring that all words are separated by a single space.
   - Removing spaces before punctuation (e.g., commas, periods).
   - Ensuring there is exactly one space after punctuation marks.
2. Count the number of words in the paragraph.
3. Find and output the longest word in the paragraph.

Input:

- A single paragraph of text as a string input (without newline characters).

Output:

- The normalized paragraph.
- The total word count.
- The longest word in the paragraph.

  If there are multiple words of the same length, output the first one.

Example:

Input	**Output**

`␣Hello␣␣␣world␣!␣␣This␣is␣␣a␣␣test␣␣.␣␣`	`Normalized paragraph:`
	`Hello␣world!␣This␣is␣a␣test.`
	`Word count:  6`
	`Longest word:  Hello`

Notes: The input string length does not exceed 10^5 characters and consists of only printable ASCII characters.

Exercise 4: Linked List

Implement a singly linked list in Python with the following structure:

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

Implement the following operations:

1. **Traversal**: Print the linked list.
2. **Count Nodes**: Return the total number of nodes.
3. **Add Head**: Insert a node at the beginning.
4. **Add Tail**: Insert a node at the end.
5. **Remove Head**: Delete the first node.
6. **Remove Tail**: Delete the last node.
7. **Remove Duplicates**: Remove duplicate elements.

Exercise 5: Stack

Using the linked list implementation above, implement a stack with the following operations:

1. **Push**: Add an item to the stack.
2. **Pop**: Remove the top item.
3. **Top**: Get the top item without removing it.

Exercise 6: Queue

Using the linked list implementation above, implement a queue with the following operations:

1. **Enqueue**: Add an item to the queue.
2. **Dequeue**: Remove the front item.
3. **Front**: Get the front item without removing it.

Regulations

Please follow these guidelines:

- You may use any Python IDE.
- After completing assignment, check your submission before and after uploading to Moodle.
- Do not use the following modules: `numpy`, `pandas`, `collections`, `heapq`, and `deque`.
- You may use `list`, `tuple`, and `set` but no external libraries.

Your submission must be contributed in a compressed file, named in the format `StudentID.zip`, with the following structure:

```
StudentID
 ├─ Exercise_1.py
 ├─ Exercise_2.py
 ├─ Exercise_3.py
 ├─ Exercise_4.py
 ├─ Exercise_5.py
 └─ Exercise_6.py
```

The end.