Previous Lab Questions

ISS TAs

1 Part I: NumPy Programming

1. The Threshold Replacer

Problem Statement:
You are given a 1D NumPy array representing sensor data. Some readings are considered "noise" if they fall strictly below a certain threshold T. Your task is to filter this noise by replacing all values strictly less than T with the **mean** of the original array.

Input Format:

- First line: A list of space-separated integers representing the array elements.
- Second line: An integer T (the threshold).

Output Format:

- Print the modified array rounded to 2 decimal places.

Constraints:

- Do not use explicit loops (must use NumPy masking/vectorization).

Sample Input

```
10 2 30 4 50
15
```

Sample Output

```
[19.2, 19.2, 30.0, 19.2, 50.0]
```

2. Matrix Standardization & Outlier Pruning

Problem Statement:
Given a 2D matrix (rows are samples, columns are features), perform the following steps:

1. **Standardize column-wise:** Subtract the column mean from each element and divide by the column standard deviation.
2. **Prune Outliers:** After standardization, replace any value > 1.0 or < -1.0 with 0.

Input Format:

- First line: Integers N and M (dimensions of the matrix).
- Next N lines: M space-separated floats.

Output Format:

- Print the final processed matrix rounded to 2 decimal places.

Technical Constraints:

- Ensure `axis=0` is used for mean/std calculations.
- Assume standard deviation is never 0 for test cases.

Sample Input

```
3 3
1 2 9
6 4 3
9 7 8
```

Sample Output

```
[[0.0, 0.0, 0.89], [0.2, -0.16, 0.0], [0.0, 0.0, 0.51]]
```

3. Weighted Student Ranking (Vectorized)

Problem Statement:
You are building a ranking system for a class with N students and M subjects.

1. **Weighted Average:** For each student i, compute the weighted average:
$$\text{weighted\_avg}_i = \sum_{j=0}^{M-1} \text{scores}[i][j] \times \text{weights}[j]$$
2. **Qualification Check:** A student is **qualified** only if they scored at least **35** in *every* subject.
3. **Ranking:** Rank qualified students by their weighted average (descending). If averages are tied, order by original index (ascending).
4. **Top K:** Return the indices of the top K qualified students.

Input Format:

- Line 1: Three integers N, M, K.
- Line 2: M floats representing weights.
- Next N lines: Student scores.

Output Format:

- A single line containing space-separated indices of the top K qualified students.
- If no students qualify, print an empty line.

Sample Input

```
2 2 2
0.5 0.5
40 90
30 100
```

Sample Output

```
0
```

Explanation: Student 1 has a score of 30, which is < 35, so they are disqualified.

2 Part II: Bash Scripting

1. Permission Octal Calculator

Task: Write a script that reads a 3-character string representing file permissions (e.g., `rw-`) and calculates the corresponding octal digit based on the bit values ($r = 4, w = 2, x = 1$).
Example: Input `r-x` $\rightarrow 4 + 0 + 1 = 5$.
Input/Output:

- **Input:** A single line with exactly 3 characters (`r`, `w`, `x`, or `-`).
- **Output:** A single integer (0-7).

Sample Input / Output

```
Input:  rw-
Output: 6
```

2. Log Scanner

Task: Read log data from STDIN. Search for the text "CRITICAL" (case-insensitive).

- If found, output: `Danger Found`
- If NOT found, output: `All systems nominal`

Note: Do not output the actual log lines.

Sample Input

```
[INFO] Boot
[WARN] CPU High
[CRITICAL] DB Fail
```

Sample Output

```
Danger Found
```

3. Log Splitter & Reporter

Task: Read a Session ID from the first line of input. Read the rest of the stream as logs.

1. **File Creation:** Create two files based on the Session ID:
   - `<Session_ID>_system.log` (For INFO logs).
   - `<Session_ID>_summary.txt` (For the report).
2. **Processing:**
   - Lines containing "ERROR" → Redirect to **STDERR**.
   - Lines containing "INFO" → Append to `..._system.log`.
   - Ignore other lines (DEBUG, WARN).
3. **Reporting:** Count occurrences of the code "SHAW" (case-insensitive) inside the new system log file. Populate the summary file exactly as:

   ```
   REPORT
   Shaw Events: <Count>
   End of Report
   ```

4. Print `Archive Synced` to **STDOUT**.

3 Part III: Python Algorithms

1. The Mountain Trail

Problem Statement:
Rearrange a list of elevation measurements into a "Mountain" pattern:

1. Split the list into two halves. If n is odd, the first half gets the extra element.
2. **Ascent:** The first half must be sorted in non-decreasing order.
3. **Descent:** The second half must be sorted in non-increasing order.

Input Format: A single line of space-separated integers. ($2 \leq n \leq 10^5$).
Output Format: The rearranged sequence on a single line.

Sample Input

```
5 2 8 1 9
```

Sample Output

```
2 5 8 9 1
```

2. Lexicographical Permutations

Problem Statement:
Given a string S of unique characters, generate every possible rearrangement (permutation). Output them in lexicographically sorted (alphabetical) order.

Input Format: A single string S ($1 \leq len(S) \leq 8$).
Output Format: Space-separated strings on a single line.

Sample Input

```
zen
```

Sample Output

```
enz ezn nez nze zen zne
```

3. Two Sum (Optimized)

Problem Statement:
Given an array of integers `nums` and an integer `target`, return the indices of the two numbers such that they add up to `target`.

- You may assume exactly one solution exists.
- You may not use the same element twice.
- Return indices in ascending order.
- **Constraint:** Time complexity must be less than $O(n^2)$.

Input Format:

- First line: t (number of test cases).
- For each test case:
  - Line 1: n (size of array).
  - Line 2: n space-separated integers.
  - Line 3: The target sum.

Sample Input

```
1
4
2 7 11 15
9
```

Sample Output

```
0 1
```