Mark as done

Description Submission view

Available from: Tuesday, 23 September 2025, 9:15 AM
Due date: Tuesday, 23 September 2025, 10:45 AM
Requested files: p1.py, p2.py, p3.py, p4.py (Download)
Type of work: Individual work

Problem 1: Consecutive Equal Numbers

Description
Write a program to count how many numbers are exactly equal to the number immediately before them.

Concept
This problem tests your ability to compare adjacent elements in a list.

Task

1. Prompt:
   `Enter numbers:`
2. Read the input list of integers.
3. For each index from 1 to end, check if the number equals the previous one.
4. Count matches and print the result.

Output format

- Print: `Count: [value]`

Example 1:

```
Enter numbers: 8 8 2 2 5 5
Count: 3
```

Explanation:
List: [8, 8, 2, 2, 5, 5]
Compare each number with its previous neighbor:
- Index 1: 8 (current) equals 8 (previous) → count = 1
- Index 2: 2 (current) vs 8 (previous) → not equal
- Index 3: 2 (current) equals 2 (previous) → count = 2
- Index 4: 5 (current) vs 2 (previous) → not equal
- Index 5: 5 (current) equals 5 (previous) → count = 3
Total consecutive equal pairs: 3

Example 2:

```
Enter numbers: 10 20 20 30 30 30
Count: 3
```

Explanation:
List: [10, 20, 20, 30, 30, 30]
Compare each number with its previous neighbor:
- Index 1: 20 (current) vs 10 (previous) → not equal
- Index 2: 20 (current) equals 20 (previous) → count = 1
- Index 3: 30 (current) vs 20 (previous) → not equal
- Index 4: 30 (current) equals 30 (previous) → count = 2
- Index 5: 30 (current) equals 30 (previous) → count = 3
Total consecutive equal pairs: 3

?

Restrictions

- At least one number is given.
- If the input list has only one element, Output: `Count: 0`

Problem 2 - Product of Row Medians

Description:
Write a program to compute the product of the medians of each row in a matrix. For this problem, assume that every row in the input matrix is already sorted in non-decreasing order.

Concept:
This problem involves iterating through a matrix and performing a calculation on each row. After a check for valid matrix, the core concept is finding the median of a sorted list. This requires handling cases for when matrix either has odd or even length rows based on the problem's specific definition of the median.

Task:
Your program must produce output that exactly matches the format specified.
1. Prompt for the number of rows with: `Enter number of rows: `.
2. Prompt for the matrix data with: `Enter matrix rows with each row on a new line:`. The user will then enter each row on a separate line with elements separated by spaces.
3. Check if the matrix is valid (all rows have the same length). If not, print `-1` and stop.
4. For each row, determine its median. The median is the middle element when matrix has odd-length rows. When matrix has even-length rows, the median is defined as the smaller of the two middle elements.
5. Calculate the product of all medians found across the rows.
6. Finally, print the resulting product.

Example:
*(Text in **bold** is what the user types.)*

```
Enter number of rows: 3
Enter matrix rows with each row on a new line:
1 3 5
2 4 6 8
10 20
-1
```

(Another example)

```
Enter number of rows: 4
Enter matrix rows with each row on a new line:
10 30
20 25
15 20
5 10
15000
```

Restrictions:
Each row in the input matrix will be pre-sorted.
All matrix elements will be integers.
No external libraries are necessary.

Problem 3: Count Cells Greater Than Neighbors

Description
Write a program that takes integers as input to form a matrix.
A cell is considered **greater than its neighbors** if its value is **strictly greater than each of its existing neighbors**:

- Up (row–1, col)
- Down (row+1, col)
- Left (row, col–1)
- Right (row, col+1)

The program should count and print the total number of such cells.

Concept:
This problem tests the ability to:

- Represent and traverse 2D arrays (matrices).
- Access neighbors conditionally (checking bounds).
- Apply comparison logic to identify special cells.

Task:

1. Prompt the user to enter the number of rows(r) and columns(c).
2. Take the matrix elements as input in row-major order.
3. Make sure that the number of matrix elements = r*c, if not, print "INVALID INPUT"

4. For each cell in the matrix:
   Check all **valid neighbors** (ignore neighbors that fall outside the matrix).
   If the cell is strictly greater than all of its neighbors, count it.

Input format

```
Enter number of rows and columns: <Two space separated integers>
Enter matrix elements: <space-separated integers>
```

Output format

```
<value>(if number of matrix elements = r*c)
INVALID INPUT(if number of matrix elements != r*c)
```

Example 1:

Input:

```
Enter number of rows and columns: 3 3
Enter matrix elements: 1 2 1 4 5 6 3 2 1
```

Output:

```
1
```

Explanation:
Matrix:
1 2 1
4 5 6
3 2 1

The only cell greater than all 4 neighbors is **6**

Example 2:

Input:

```
Enter number of rows and columns: 2 3
Enter matrix elements: 1 9 2 3 5 4
```

Output:

```
1
```

Explanation:
Matrix:
1 9 2
3 5 4

The only cell greater than all 4 neighbors is **9**

Restrictions

- All values should be integers. Row, column >1
- No external libraries.

Problem 4 - Reverse a List In-Place

Description:
Given a list of integers, write a program to reverse its elements without creating a new list. This is known as an "in-place" reversal.

Concept:
This problem tests a fundamental algorithm for in-place data manipulation. Since you cannot create a new list, you must modify the existing one directly to reverse its contents. This requires careful management of indices to alter the list element by element without using extra memory for a second list.

Important: New Workflow for this Problem
For this problem, you have been provided with a template file, `p4.py`, and a special utility library called `lab_utils`. You must use this template and its helper functions to complete the task.

Task:
Your task is to complete the logic in the provided `p4.py` template file.

1. The template already handles getting input from the user and creating a list named `numbers`.
2. You must write Python code in the designated section of the template to modify the `numbers` list in-place.
3. The template will then automatically handle printing your final, modified list.

Using the Helper Functions (from `lab_utils`)
You do not need to write these functions; they are provided for you in the template.

- `get_list_from_input(...)`: This function is used at the beginning of the template to get the list from the user. You do not need to change this line.
- `submit_answer(numbers)`: At the end of the template, this function is called to print your final list. It also performs a background check to ensure you modified the list in-place.

Example:
*(The final program interaction will look like this. Text in **bold** is what the user types.)*

```
Enter a list of numbers separated by spaces: 10 20 30 40 50
[50, 40, 30, 20, 10]
```

Restrictions:

- You **must not** use the built-in `print()` function. It has been disabled for this problem. You must let the template's `submit_answer()` call handle the output.
- You **must not** use any built-in reverse functions like `.reverse()` or `reversed()`.
- You **must not** use slicing to reverse the list (e.g., `my_list[::-1]`).
- You **must not** create a new list to store the elements. The modification must happen "in-place" on the original `numbers` list.
- You should only add code to the designated section in the `p4.py` template. Do not change the existing lines.

Requested files

p1.py

```
# write your code here below
```

p2.py

```
# write your code here below
```

p3.py

```
# write your code here below
```

p4.py

```python
# p4.py (Student Template)

# You must use these functions from the lab_utils library.
# Do not use the built-in 'print' function for this problem.
from lab_utils import get_list_from_input, submit_answer

# This line gets the input and prepares the list. Do not change it.
numbers = get_list_from_input("Enter a list of numbers separated by spaces: ")

# --- YOUR CODE GOES HERE ---
# Write your code to reverse the 'numbers' list in-place.
# Do not create a new list. Modify the 'numbers' list directly.
# For example:
# left = 0
# right = len(numbers) - 1
# ... your swapping logic ...










# --- END OF YOUR CODE ---

# This line will check your work and print the result. Do not change it.
submit_answer(numbers)
```

VPL