Mark as done

Description Submission view

Available from: Monday, 22 September 2025, 9:15 AM
Due date: Monday, 22 September 2025, 10:45 AM
Requested files: p1.py, p2.py, p3.py, p4.py (Download)
Type of work: Individual work

Problem 1: Consecutive Numbers with Different Parity

Description
Write a program that counts how many times two consecutive numbers in a list have different parity (i.e., one is even and the other is odd).

Concept
This problem checks your ability to traverse lists using loops and apply conditional logic based on parity (even/odd).

Task

1. Prompt the user:
   `Enter numbers:`
2. Read a line of space–separated integers and store them in a list.
3. Traverse the list, comparing each element with the next one.
4. Count pairs where one is even and the other is odd.
5. Print the count.

Input format

- User enters space-separated integers in one line. After the prompt:
  `Enter numbers:`

Output format

- Print: `Count: [value]`

Example 1:

```
Enter numbers: 2 3 4 7 8

Count: 4
```

Explanation:

- List: [2, 3, 4, 7, 8]
- Consecutive pairs:
  - (2, 3): even (2) and odd (3) → different parity → count = 1
  - (3, 4): odd (3) and even (4) → different parity → count = 2
  - (4, 7): even (4) and odd (7) → different parity → count = 3
  - (7, 8): odd (7) and even (8) → different parity → count = 4
- **Total pairs with different parity: 4** (so output is 4)

Example 2:

```
Enter numbers: 1 3 5 7

Count: 0
```

Explanation:

?

- List: [1, 3, 5, 7] (all odd numbers)
- Consecutive pairs:
  - (1, 3): odd and odd → same parity → no count
  - (3, 5): odd and odd → same parity → no count
  - (5, 7): odd and odd → same parity → no count
- **Total pairs with different parity: 0** (so output is 0)

Restrictions

- No external libraries.
- Input contains at least two integers.

Problem 2 - Symmetric Pair Counter in a Matrix

Description:
Write a program that takes a matrix (a nested list of numbers) and counts the number of "symmetric pairs." A symmetric pair is a pair of elements `a[i][j]` and `a[j][i]` that are equal. Your count should only include pairs where `i` is not equal to `j` (i.e., ignore elements on the main diagonal).

Concept:
This problem requires you to first validate that the input is a valid matrix. The core of the problem involves using nested loops to access elements by their indices, `(i, j)`, and checking for symmetry *where possible* within the matrix bounds, even if it is not square. You must devise a strategy to count each symmetric pair only once.

Task:
Your program must produce output that exactly matches the format specified.
1. Prompt the user for the number of rows with the message: `Enter number of rows: `.
2. Prompt the user for the matrix data with the message: `Enter matrix rows with each row on a new line:`. The user will then enter each row on a separate line with its elements separated by spaces.
3. Check if the matrix is valid (all rows have the same length). If not, print `-1` and stop.
4. Initialize a counter variable to zero.
5. Iterate through the matrix to find and count all symmetric pairs `(a[i][j] == a[j][i])`. Ensure your logic can handle non-square matrices by only checking indices that are valid for both `a[i][j]` and `a[j][i]`.
6. After the loops complete, print the final count.

Warning:
A common pitfall is double-counting. A standard nested loop structure that iterates through all `(i, j)` indices will naturally check pairs in both directions. For example, if your code confirms a pair at indices `(1, 3)` by checking `a[1][3] == a[3][1]`, it will eventually also check the pair at `(3, 1)`. This is the same symmetric pair and must only be counted once.

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 2 3
2 4 5
3 6 7
2
```

(Another example)

```
Enter number of rows: 2
Enter matrix rows with each row on a new line:
10 20
50 60 70
-1
```

Restrictions:
All matrix elements will be integers.
No external libraries are necessary.

Problem 3: Diagonal Difference in a Matrix

Description:

Write a program that takes as input a matrix of integers and computes the absolute difference between the sums of its main and secondary diagonals.

- If it is square, compute:
  - The main diagonal sum (top-left → bottom-right).
  - The secondary diagonal sum (top-right → bottom-left).
  - Print the absolute difference between these two sums.
- If it is not square, i.e. r ≠ c , print `NOT A SQUARE MATRIX`.
- If the number of elements provided is not exactly `row*column`, print `INVALID INPUT`

Concept:

This problem tests the ability to:

- Handle 2D arrays (matrices).
- Check if a matrix is square.
- Traverse diagonals of a square matrix.

Task:

1. Prompt the user with:
   `Enter number of rows and columns:`
2. Prompt the user with:
   `Enter matrix elements:`
3. If r ≠ c, print `"NOT A SQUARE MATRIX"` and stop.
4. If the number of elements provided is not exactly `r*c`, print `INVALID INPUT` and stop.
5. Otherwise:
   - Calculate the main diagonal sum (top-left → bottom-right).
   - Calculate the secondary diagonal sum (top-right → bottom-left).
   - Print the absolute difference between these sums:
     `<value>`

Input format

```
Enter number of rows and columns: <two space-separated integers>
Enter matrix elements: <r*c space-separated integers>
```

Output format

If the matrix is not square:

```
NOT A SQUARE MATRIX
```

If the number of elements is incorrect:

```
INVALID INPUT
```

If input is valid:

```
<absolute difference>
```

Example 1 (Square Matrix, Valid Input):

```
Enter number of rows and columns: 3 3
Enter matrix elements: 11 2 4 4 5 6 10 8 -12
15
```

Explanation:

- Main diagonal = 11 + 5 + (-12) = 4
- Secondary diagonal = 4 + 5 + 10 = 19
- Absolute difference = |4 - 19| = 15

Example 2 (Non-Square Matrix → NOT A SQUARE MATRIX):

```
Enter number of rows and columns: 2 3
NOT A SQUARE MATRIX
```

Example 3 (Invalid Input — Wrong Number of Elements):

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

Explanation:

- Rows = 3, Columns = 3 → Expected `3*3 = 9` elements.
- User entered only 5 elements.
- Hence, output is `INVALID INPUT`.

Restrictions

- All values should be integers.
- `r, c > 1`.
- Matrix elements can be positive or negative integers.
- No external libraries or predefined matrix function allowed.

Problem 4 - Longest Consecutive Run of a Character

Description:
Given a string, find the character that has the longest "run" of consecutive appearances. Print the character and the length of its run. If multiple characters have runs of the same maximum length, choose the one whose run appears first in the string.

Concept:
This is a classic string iteration problem that requires tracking state as you scan the string. You will need to identify consecutive runs of identical characters, measure their lengths, and keep track of the longest run encountered according to the problem's rules.

Task:
Your program must produce output that exactly matches the format specified.
1. Prompt the user for a string with the message: `Enter a string:` .
2. Initialize any necessary variables to track the longest run found so far.
3. Iterate through the string to identify the character and length of the longest consecutive run. Ensure your logic correctly handles the tie-breaking rule by selecting the run that appears first in the string.
4. After processing the entire string, print the final character and length of the longest run, separated by a space.

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

```
Enter a string: aabbbccccddeeeeee
e 6
```

(Another example)

```
Enter a string: konoohaa
o 2
```

Restrictions:
The input string will be non-empty.
You should solve this by iterating through the string manually.

Requested files

p1.py

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

p2.py

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

p3.py

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

p4.py

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

VPL