Mark as done

Description Submission view

Available from: Friday, 26 September 2025, 9:15 AM
Due date: Friday, 26 September 2025, 10:45 AM
Requested files: p1.py, p2.py, p3.py, p4.py (Download)
Type of work: Individual work

Problem 1: Odd Numbers at Even Indices

Description
Write a program that counts how many numbers in a list are **odd** and located at an **even index** (0, 2, 4, ...).

Concept
This combines **index checking** (even index) with a **number property** (odd).

Task

1. Prompt the user:
   `Enter numbers:`
2. Read space–separated integers into a list.
3. Traverse only even indices.
4. For each, check if the number is odd.
5. Count and print the total.

Input format

- A line of space–separated integers.

Output format

- Print:
  `[value]`

Example 1

```
Enter numbers: 1 2 3 4 5 6
4
```

Explanation:

- List: [1, 2, 3, 4, 5, 6] (indices 0 to 5)
- Check even indices (0, 2, 4):
  - Index 0: value 1 (odd) → count = 1
  - Index 2: value 3 (odd) → count = 2
  - Index 4: value 5 (odd) → count = 3
- Total: 3

Example 2

```
Enter numbers: 2 4 6 8 10
0
```

Explanation

- List: [2, 4, 6, 8, 10] (all even numbers at even indices)
- Check even indices (0, 2, 4):
  - Index 0: value 2 (even) → no count
  - Index 2: value 6 (even) → no count
  - Index 4: value 10 (even) → no count
- Total: 0

?

Restrictions

- Input list length ≥ 1.
- Use 0 based indexing (i.e. Indexing starts from 0).
- No external libraries or predefined functions allowed.

Problem 2 - Row with Largest Sum

Description:
Write a program that finds the row in a matrix that has the largest sum of elements. Your program should print the row number of this row (***starting from 1***). If there's a tie for the largest sum, the row with the smallest row number should be chosen.

Concept:
This problem involves iterating through the matrix and calculating the sum of each row. The core concept is tracking a running maximum. As you iterate and calculate sums, you must compare each sum to the largest sum found so far and update your result accordingly, while correctly handling the tie-breaking rule. The final output must be converted from a 0-based list index to a 1-based row number.

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 rectangular. If not, print `-1` and stop.
4. Calculate the sum of each row.
5. Identify the row with the largest sum, adhering to the tie-breaking rule (smallest row number wins).
6. Print the row number of this row. Note that row numbers are 1-indexed (the first row is row 1, the second is row 2, and so on).

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

```
Enter number of rows: 4
Enter matrix rows with each row on a new line:
1 2 3
10 0 1
5 5 1
4 4 2
2
```

(Explanation: Row 1 sum=6. Row 2 sum=11. Row 3 sum=11. Row 4 sum=10. The largest sum is 11, which occurs first in row 2. Therefore, the answer is 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:

- Rows are numbered from 1 for output.
- All matrix elements will be integers.
- No external libraries are necessary.

Problem 3: Count Saddle Points in a Matrix

Description:
You are given a matrix of integers. A **saddle point** in a matrix is defined as an element that is:

- The **minimum element** in its row, **and**
- The **maximum element** in its column.

Your task is to count how many saddle points exist in the given matrix. If no saddle points are present, output `0`.

Concept:
This problem tests the ability to:

- Traverse a matrix row by row and column by column.
- Compare elements across rows and columns simultaneously.
- Implement combined conditions to detect special elements (saddle points).

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, i.e. all elements in a single line, with space between each element.
3. Make sure that the number of matrix elements = r*c, if not, print "INVALID INPUT"
4. For each element, check if it is the minimum in its row and maximum in its column.
5. Count the total number of such saddle points.
6. Print the count.

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: 4 5 6 7 8 9 1 2 3
```

Output:

```
1
```

Explanation:
Matrix:
4 5 6
7 8 9
1 2 3

Row minimums = [4, 7, 1] Column maximums = [7, 8, 9]
Only element 7 is both the minimum in row 2 and maximum in column 1 → Count = 1

Example 2:

Input:

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

Output:

```
1
```

Explanation:
Matrix:
1 2
3 4

Row minimums = [1, 3] Column maximums = [3, 4]
Element 3 is simultaneously the row-minimum and column-maximum → Count = 1

Example 3:

Input:

```
Enter number of rows and columns: 3 3
Enter matrix elements: 4 5 6 7 8 9 7 2 3
```

Output:

```
1
```

Explanation:
Matrix:
4 5 6
7 8 9
7 2 3

Row minimums = [4, 7, 2] Column maximums = [7, 8, 9]
Only element 7(row=2, col=1) is the minimum in it's respective rows and maximum in it's respective columns → Count = 1

Restrictions

- All values should be integers. Row, column >1, matrix elements >= 0
- No external libraries or predefined functions allowed.

Problem 4 - Custom Split Function

Description:
Write a program that mimics the behavior of the `.split()` string method. Given a main string `s` and a delimiter string `sep`, split `s` into a list of substrings. The split should occur at every occurrence of `sep`.

Concept:
This problem involves string parsing and list construction. The core logic is to repeatedly locate the delimiter `sep` in the string and use its position to define the boundaries of the substrings. Careful handling of edge cases, such as consecutive or terminal delimiters, is required to produce the correct list of substrings, including empty strings.

Task:
Your program must produce output that exactly matches the format specified.
1. Prompt for the main string with: `Enter main string:` .
2. Prompt for the delimiter with: `Enter separator:` .
3. Create a list of substrings by splitting the main string `s` at every occurrence of the delimiter `sep`.
4. Your implementation must correctly handle all cases, including consecutive delimiters (which produce empty strings), and delimiters at the beginning or end of the string.
5. Print the final list of substrings.

Example 1:
*(Text in **bold** is what the user types.)*

```
Enter main string: a,,b,c
Enter separator: ,
['a', '', 'b', 'c']
```

Example 2: *Using space as the delimiter*

```
Enter main string: hello world python
Enter separator:
['hello', 'world', 'python']
```

Restrictions:
You must not use the built-in `.split()`, `.rsplit()` or other string methods.

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