Mark as done

Description Submission view

Available from: Friday, 17 October 2025, 9:15 AM
Due date: Friday, 17 October 2025, 10:45 AM
Requested files: p1.py, p1_input.txt, p2.py, p2_input.txt, p3.py, p3_input.txt, p4.py, p4_input.txt (Download)
Type of work: Individual work

Instructions for LabTest3-Day5

This set contains 4 problems, the descriptions of which are provided below. A starter code is provided for each problem: p1.py, p2.py, p3.py, and p4.py for problems 1,2,3 and 4, respectively. Each file contains some sections that are necessary for code execution and an editable section where you are expected to write your code.

Do not change anything in the non-editable sections; otherwise, you will run into issues.

Running test cases:

You are provided a text file named pno_input.txt for each problem. You can use this file to test your code for any errors. The inputs for each problem can be provided in this file. Sample test cases are provided for all problems. You can add as many testcases you want in corresponding input files. *Usage of each input file is described in the corresponding problem description.*

Problem 1: Rotate Matrix 90 Degrees

Description:

Create a Python function named *rotate_matrix_90* that rotates a given square matrix (list of lists) 90 degrees. The function should support both clockwise and counter-clockwise rotation.

Function Definition:

You will define the function with the following signature:

```
def rotate_matrix_90(matrix: List[List[Any]], clockwise: bool = True) -> List[List[Any]]:
```

- **Input parameters:**
  - *matrix*: A list of lists representing a square matrix.
  - *clockwise*: A boolean, defaulting to True.
    - If True, rotate clockwise.
    - If False, rotate counter-clockwise.
- **Output parameter:** A new list of lists representing the rotated matrix.

Input File (p1_input.txt):

Your solution will be tested using input provided in the p1_input.txt file. You should update this file to contain the desired test cases.

- **Input file format:**
  - The first line indicates the number of test cases.
  - Each subsequent test case will consist of:
    - A line with the boolean clockwise value (True or False).
    - A line with the size of the square matrix (N).
    - N lines, each representing a row of the matrix, with elements separated by spaces. Elements can be integers or strings.
- **Example p5_input.txt content:**

```
2
True
3
1 2 3
4 5 6
7 8 9
False
2
A B
C D
```

Expected Output Example 1 (Clockwise):

Given matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] and clockwise = True

Your function should produce the output:

```
[[7, 4, 1], [8, 5, 2], [9, 6, 3]]
```

Expected Output Example 2 (Counter-clockwise):

Given matrix = [['A', 'B'], ['C', 'D']] and clockwise = False

Your function should produce the output:

```
[['B', 'D'], ['A', 'C']]
```

Notes and Restrictions:

- The input matrix will always be a square matrix (N x N).
- Your function should return a new matrix. Do not modify the original input matrix in-place.
- Elements in the matrix can be integers or strings. Ensure your solution handles generic Any type elements.
- You will NOT need any predefined fucntions or libraries.

Problem 2: Filter and Transform List

Description:

Write a function named ***filter_and_transform*** that takes a *list of items*, a *filter_func*, and a *transform_func*.

The function should first apply the *filter_func* to each item in the list. Only items for which *filter_func* returns True should be kept. Then, for each of the filtered items, apply the *transform_func* and collect the results into a new list.

The function should return this new list of transformed items. The order of items in the output list should preserve their relative order from the original input list after filtering. All the helper functions (*filter_func*) are already defined, and you should not edit the definition of these helper functions.

Function Definition:

You will define the function with the following signature:

```
def filter_and_transform(items: List[Any], filter_func: Callable[[Any], bool], transform_func: Callable[[Any], Any]) -> List[Any]:
```

- **Input parameters:**
  - items: A list of any type of elements.
  - filter_func: A callable function that takes an item and returns True or False.
  - transform_func: A callable function that takes an item and returns its transformed version.
- **Output parameter:** A new list containing the transformed items that passed the filter.

Input File (p2_input.txt):

Your solution will be tested using input provided in the p2_input.txt file. You should update this file to contain the desired test cases.

- **Input file format:**
  - The first line indicates the number of test cases.

- Each test case will consist of:
  - A line with the name of the filter_func (as a string).
  - A line with the name of the transform_func (as a string).
  - A line with the number of items in the list.
  - Subsequent lines, each representing an item of the list (can be integers or strings).

- **Example p2_input.txt :**

```
2
is_even
square_it
5
1
2
3
4
5
has_char_e
to_uppercase
3
apple
banana
grape
```

Expected Output Example 1:

Given items = [1, 2, 3, 4, 5], filter_func = is_even, transform_func = square_it

1. **Filter:** [2, 4] (since is_even(2) is True, is_even(4) is True)
2. **Transform:**
   1. square_it(2) -> 4
   2. square_it(4) -> 16

Your function should return:

```
[4, 16]
```

Notes and Restrictions:

- Ensure that filtering happens before transformation.
- The relative order of elements from the original list (after filtering) must be maintained in the output.
- The filter_and_transform function should return a new list, not modify the input list.
- No external libraries needed.

Problem 3 - Recursive Digit Sum

Description:

Create a Python function named `digit_sum` that takes a single positive integer `n` and returns the **sum of all its digits**.

The function must be implemented **recursively**, without using loops (`for`, `while`) or built-in functions like `sum()` on lists of digits.

Function Definition:
You will define the function with the following signature:
`def digit_sum(n: int) -> int:`

Input parameters:

- `n`: A positive integer whose digits need to be summed.

Output parameter:

- An integer representing the sum of all digits in `n`.

Input File (p3_input.txt):

Your solution will be tested using input provided in the `p3_input.txt` file. You should update this file to include your desired test cases.

Input file format:

The first line indicates the number of test cases.

Each subsequent line contains a single integer `n`.

Example p3_input.txt content:

```
3
1234
567
908
```

Output:

```
10
18
17
```

Notes and Restrictions:

- You must implement the function **recursively**.
- Do **not** use loops (`for`, `while`) or built-in functions that directly compute the sum of digits.
- You can use integer division (`//`) and modulo (`%`) to extract digits.
- Assume all input numbers are non-negative integers, i.e. n > 0

Problem 4: Longest Subarray with Equal Zeros and Ones

Description:

Create a Python function named `find_longest_subarray` that takes a binary array (a list containing only 0s and 1s). The function should find the length of the longest contiguous subarray that has an equal number of 0s and 1s.

Function Definition:

```
def find_longest_subarray(nums: List[int]) -> int:
```

- **Parameters:**
  - `nums`: A list of integers containing only 0s and 1s.
- **Returns:** An integer representing the length of the longest contiguous subarray with an equal number of 0s and 1s.

Input File (`p4_input.txt`):

- The first line is the number of test cases.
- Each subsequent line contains a list of 0s and 1s separated by spaces.

Example `p4_input.txt`:

```
2
0 1 0 0 1 1 0
1 1 1 1
```

Expected Output for Example:

For the first case, `[0, 1, 0, 0, 1, 1]` is the longest subarray with three 0s and three 1s. For the second case, no such non-empty subarray exists.

```
6
0
```

Notes:

- If no such subarray exists, the function should return 0.
- The input list will only ever contain the integers 0 and 1.

Requested files

p1.py

```python
from typing import List, Any

def rotate_matrix_90(matrix: List[List[Any]], clockwise: bool = True) -> List[List[Any]]:
    # write code here








############################ Do Not Change #################################

def solution(tc):
    inp_matrix, inp_clockwise = tc
    return rotate_matrix_90(inp_matrix, inp_clockwise)


def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines]
    num_tests = int(lines[0])
    input_tests = []
    current_line_idx = 1

    for _ in range(num_tests):
        clockwise_str = lines[current_line_idx]
        clockwise_bool = True if clockwise_str.lower() == 'true' else False
        current_line_idx += 1

        n = int(lines[current_line_idx])
        current_line_idx += 1

        matrix = []
        for _ in range(n):
            row_str = lines[current_line_idx].split()
            row = []
            for x in row_str:
                if x.isdigit() or (x.startswith('-') and x[1:].isdigit()):
                    row.append(int(x))
                else:
                    row.append(x)
            matrix.append(row)
            current_line_idx += 1

        input_tests.append((matrix, clockwise_bool))

    return input_tests


if __name__ == "__main__":
    Input = process_input('p1_input.txt')
    for tc in Input:
        print(solution(tc))

```

p1_input.txt

```
2
True
3
1 2 3
4 5 6
7 8 9
False
2
A B
C D
```

p2.py

```python
def filter_and_transform(items, filter_func, transform_func):
    #write code here








############################ Do Not Change #################################

# Example helper functions - these will be available to the VPL system's process_input
# Students do NOT need to implement these.
def _is_even(x):
    return x % 2 == 0

def _is_positive(x):
    return x > 0

def _square_it(x):
    return x * x

def _double_it(x):
    return x * 2

def _has_char_e(s):
    if isinstance(s, str):
        return 'e' in s.lower()
    return False

def _to_uppercase(s):
    if isinstance(s, str):
        return s.upper()
    return s

_func_map = {
    "is_even": _is_even,
    "is_positive": _is_positive,
    "square_it": _square_it,
    "double_it": _double_it,
    "has_char_e": _has_char_e,
    "to_uppercase": _to_uppercase,
}

def solution(tc):
    items, filter_func_name, transform_func_name = tc
    filter_func = _func_map[filter_func_name]
    transform_func = _func_map[transform_func_name]
    return filter_and_transform(items, filter_func, transform_func)

def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines]
    num_tests = int(lines[0])
    input_tests = []
    current_line_idx = 1

    for _ in range(num_tests):
        filter_name = lines[current_line_idx]
        current_line_idx += 1

        transform_name = lines[current_line_idx]
        current_line_idx += 1

        num_items = int(lines[current_line_idx])
        current_line_idx += 1

        current_items_list = []
        for _ in range(num_items):
            item_str = lines[current_line_idx]
            try:
                current_items_list.append(int(item_str))
            except:
                current_items_list.append(item_str)
            current_line_idx += 1

        input_tests.append((current_items_list, filter_name, transform_name))

    return input_tests

if __name__ == "__main__":
    Input = process_input('p2_input.txt')
    i = 0
    while i < len(Input):
        print(solution(Input[i]))
        i += 1

```

p2_input.txt

```
2
is_even
square_it
5
1
2
3
4
5
has_char_e
to_uppercase
3
apple
banana
grape
```

p3.py

```python
def digit_sum(n: int) -> int:
    # write your code here below








########################### Do Not Change #################################

def solution(inp):
    return digit_sum(inp)


def process_input(filename):
    with open(filename, 'r') as file:
        lines = [line.strip() for line in file.readlines()]

    num_tests = int(lines[0])
    input_tests = [int(lines[i]) for i in range(1, num_tests + 1)]

    return input_tests


if __name__ == "__main__":
    Input = process_input('p3_input.txt')
    for inp in Input:
        print(solution(inp))

```

p3_input.txt

```
3
1234
567
908
```

p4.py

```python
from typing import List

def find_longest_subarray(nums: List[int]) -> int:
    # write code here














########################### Do Not Change #################################

def solution(tc):
    return find_longest_subarray(tc)

def process_input(filename):
    lines = open(filename, 'r').readlines()
    lines = [line.strip() for line in lines]
    num_tests = int(lines[0])
    input_tests = []

    for i in range(1, num_tests + 1):
        if lines[i]: # Handle empty lines
            num_list = [int(x) for x in lines[i].split()]
        else:
            num_list = []
        input_tests.append(num_list)

    return input_tests

if __name__ == "__main__":
    Input = process_input('p4_input.txt')
    for num_list in Input:
        print(find_longest_subarray(num_list))
```

p4_input.txt

```
2
0 1 0 0 1 1 0
1 1 1 1
```

VPL