Mark as done

Description Submission view

Available from: Wednesday, 15 October 2025, 9:15 AM
Due date: Wednesday, 15 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-Day3

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: Group Objects by Property

Description:

Create a Python function named group_by that takes two arguments:

1. array: A list of dictionaries (objects).
2. key: A string representing a property name (a key within the dictionaries).

The function should return a new dictionary with the following structure:

- **Keys:** The unique values found for the specified key property across all objects in the input array.
- **Values:** A list of dictionaries (objects) that share that particular key value. The order of objects within these lists should be maintained as they appeared in the original array.

Function Definition:

You will define the function with the following signature:

```
def group_by(array: List[Dict[str, Any]], key: str) -> Dict[Any, List[Dict[str, Any]]]:
```

- **Input parameters:**
  - array: A list of dictionaries.
  - key: A string representing the property name to group by.
- **Output parameter:** A dictionary with grouped objects.

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 key string.

?

  - A line with the number of objects in the array.
  - Subsequent lines, each representing an object as a JSON-like string (e.g., {'id': 1, 'category': 'A'}).

- **Example p1_input.txt content:**

```
2
category
4
{"id": 1, "category": "A", "value": 10}
{"id": 2, "category": "B", "value": 20}
{"id": 3, "category": "A", "value": 30}
{"id": 4, "category": "C", "value": 40}
status
3
{"task": "Buy groceries", "status": "pending"}
{"task": "Do laundry", "status": "completed"}
{"task": "Pay bills", "status": "pending"}
```

- **Expected Output Example:**

```
{'A': [{'id': 1, 'category': 'A', 'value': 10}, {'id': 3, 'category': 'A', 'value': 30}], 'B': [{'id': 2, 'category': 'B',
'value': 20}], 'C': [{'id': 4, 'category': 'C', 'value': 40}]}
```

Notes and Restrictions:

- If an object does not have the specified key, it should be ignored.
- The order of items within the value lists (the lists of dictionaries) must be preserved according to their original appearance in the input array.

Problem 2: Function Composition

Description:

Write a higher-order function named compose that takes an arbitrary number of functions as arguments (**functions*). This compose function should return a new, composed function. The composed function, when called with an initial integer value, should apply the input functions **from right to left**. All functions will work with integers. All the helper functions are already defined, and you should not edit the definition of these helper functions.

Function Definition:

```
def compose(*functions: Callable[[int], int]) -> Callable[[int], int]:
```

- **Input parameters:**
  - *compose*: **functions* (an arbitrary number of callable functions). Each function takes an integer and returns an integer.
- **Output parameter:**
  - *compose*: A new callable function that takes an integer and returns an integer.

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 a list of function names (strings), separated by spaces, representing the functions to compose. These names will map to helper functions (like _doubler, _add_ten, _squarer from previous problems, or new ones).
    - A line with the initial integer value to apply the composed function to.
- **Example p2_input.txt content:**

```
2
doubler add_ten squarer
2
```

```
add_ten doubler
5
```

Expected Output Example 1:

Given functions *doubler*, *add_ten*, *squarer* and value = 2

The composed_func = compose(doubler, add_ten, squarer) should perform:

1. *squarer(2)* -> 4 (Rightmost applied first)
2. *add_ten(4)* -> 14
3. *doubler(14)* -> 28

Your combined output for *composed_func(2)* should be:

```
28
```

Notes and Restrictions:

- The compose function should be generic enough to work with any number of functions.
- All functions (including the input value and final output) will be integers.
- Ensure the functions are applied in **right-to-left** order.
- No external libraries needed.

Problem 3 - Recursive String Reversal

Description:

Create a Python function named `reverseString` that takes a single string `s` and returns a **new string** which is the **reverse** of `s`. The function must be implemented **recursively**, without using loops (`for`, `while`) or built-in string reversal methods such as slicing (`[::-1]`) or `reversed()`.

Function Definition:
```
def reverseString(s: str) -> str:
```

Input parameter:
`s`: A string that needs to be reversed.

Output parameter:
A new string that is the reverse of the input string.

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 test case — a string `s`.

Example p3_input.txt content:

```
3
hello
recursion
python
```

Output:

```
olleh
noisrucer
nohtyp
```

Notes and Restrictions:

- The function **must** use recursion to reverse the string.
- Do **not** use slicing (`[::-1]`), loops, or built-in reverse functions.
- Use only 1 base case to terminate the string.

Problem 4: First Non-Repeating Element

Description:

Create a Python function named `find_first_non_repeating` that takes a list of integers. The function should find and return the first element in the list that does not repeat itself anywhere else in the list. If all elements in the list appear more than once, the function should return `-1`.

Function Definition:

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

- **Parameters:**
  - `nums`: A list of integers.
- **Returns:** The first non-repeating integer, or `-1` if no such element exists.

Input File (`p4_input.txt`):

- The first line is the number of test cases.
- Each subsequent line contains a list of integers separated by spaces.

Example `p4_input.txt`:

```
2
2 3 5 3 2 6 5
10 20 10 20 30 30
```

Expected Output for Example:

```
6
-1
```

Notes:

- The "first" non-repeating element is the one that appears earliest in the original list order. For example, in `[1, 2, 1, 3]`, both `2` and `3` are non-repeating, but `2` appears first, so the answer is `2`.

Requested files

p1.py

```python
def group_by(array, key):
    #write code here










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

def solution(inputs):
    inp_arr, inp_key = inputs
    return group_by(inp_arr, inp_key)

def parse_object(line):
    line = line.strip()[1:-1]
    parts = line.split(",")
    obj = {}
    for part in parts:
        if ":" not in part:
            continue
        key, val = part.split(":", 1)
        key = key.strip().strip("'").strip('"')
        val = val.strip().strip("'").strip('"')
        if val.isdigit():
            val = int(val)
        obj[key] = val
    return obj


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):
        key = lines[current_line_idx]
        current_line_idx += 1

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

        objects_list = []
        for _ in range(num_objects):
            obj_str = lines[current_line_idx]
            obj = parse_object(obj_str)
            objects_list.append(obj)
            current_line_idx += 1

        input_tests.append((objects_list, key))

    return input_tests


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

```

p1_input.txt

```
2
category
4
{"id": 1, "category": "A", "value": 10}
{"id": 2, "category": "B", "value": 20}
{"id": 3, "category": "A", "value": 30}
{"id": 4, "category": "C", "value": 40}
status
3
{"task": "Buy groceries", "status": "pending"}
{"task": "Do laundry", "status": "completed"}
{"task": "Pay bills", "status": "pending"}
```

p2.py

```python
def compose(*functions):
    # write code here










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

# Example helper functions
def doubler(x):
    return x * 2

def add_ten(x):
    return x + 10

def squarer(x):
    return x * x

def negate(x):
    return -x

def increment(x):
    return x + 1

# Map string names to actual helper functions
_func_map = {
    "doubler": doubler,
    "add_ten": add_ten,
    "squarer": squarer,
    "negate": negate,
    "increment": increment,
}

def solution(tc):
    func_names, value = tc
    functions_to_compose = [_func_map[name] for name in func_names]
    composed_func = compose(*functions_to_compose)
    return composed_func(value)


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):
        func_names_str = lines[current_line_idx]
        func_names = func_names_str.split()
        current_line_idx += 1

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

        input_tests.append((func_names, value))

    return input_tests


if __name__ == "__main__":
    Input = process_input('p2_input.txt')
    for func_names, value in Input:
        print(solution((func_names, value)))

```

p2_input.txt

```
2
doubler add_ten squarer
2
add_ten doubler
5
```

p3.py

```python
def reverseString(s: str) -> str:
    # write your code here below














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

def solution(inp):
    return reverseString(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 = lines[1:num_tests+1]  # Strings to reverse

    return input_tests


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

```

p3_input.txt

```
3
hello
recursion
python

```

p4.py

```python
from typing import List # do not remove

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







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

def solution(tc):
    return find_first_non_repeating(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):
        num_list = [int(x) for x in lines[i].split()]
        input_tests.append(num_list)

    return input_tests

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

p4_input.txt

```
2
2 3 5 3 2 6 5
10 20 10 20 30 30
```

VPL