Mark as done

Description Submission view

Available from: Monday, 13 October 2025, 9:15 AM
Due date: Monday, 13 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-Day1

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: Filter Objects by Property

Description:

Create a Python function named `filterByProperty` that takes three arguments:

1. An array (list) of objects (dictionaries).
2. A property name (as a string) which is a key in these dictionaries.
3. A value (numeric) to match against the specified property.

The function should return a new list containing only those objects from the input array where the specified property has a value that matches the given value.

Function Definition:

You will define the function with the following signature:

```
def filterByProperty(array: list, property: str, value: int) -> list:
```

- **Input parameters:**
  - array: A list of dictionaries (objects).
  - property: A string representing the key to check.
  - value: An integer representing the value to match.
- **Output parameter:** A new list containing only the matching dictionaries.

Input File (`p1_input.txt`):

Your solution will be tested using input provided in the p1_input.txt file. You may add additional test cases to this file.

- **Input file format:**
  - The first line indicates the number of test cases.
  - Each subsequent test case will consist of:
    - A line with the property name (string) and the value (integer), separated by a space.
    - 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, 'score': 90}).

?

- **Example p1_input.txt content:**

```
2
score 90
3
{"id": 1, "score": 90}
{"id": 2, "score": 85}
{"id": 3, "score": 90}
rank 2
4
{"rank": 1, "name": "Alice"}
{"rank": 2, "name": "Bob"}
{"rank": 3, "name": "Charlie"}
{"rank": 4, "name": "David"}
```

Expected Outputs:

Example 1:

Given property = `"score"`, value = `90`,

and array = `[{'id': 1, 'score': 90},`

`{'id': 2, 'score': 85},`

`{'id': 3, 'score': 90}]`

Your function should produce the output:

```
[{'id': 1, 'score': 90}, {'id': 3, 'score': 90}]
```

Example 2:

Given property=`"rank"`, value=`2`,

and array=`[{'rank':1,'name':'Alice'},`

`{'rank':2,'name':'Bob'},`

`{'rank':3,'name':'Charlie'},`

`{'rank':4,'name':'David'}]`

Your function should produce the output:

```
[{'rank': 2, 'name': 'Bob'}]
```

Notes and Restrictions:

- Ensure your function returns a new list. Do not modify the original input array.
- The value will always be numeric (integer).

Problem 2: Apply Function N Times

Description:
Create a Python function named `apply_n_times` that applies a given function (func) to an initial value n times. The result of each application becomes the input for the next. The n parameter should have a default value of 1.

Function Definition:

```
def apply_n_times(value: Any, func: Callable[[Any], Any], n: int = 1) -> Any:
```

- **Input parameters:**
  - `value:` The initial value to which the function is applied. Can be of following types: `{str, int}`.
  - `func:` The function to be applied. It takes one argument and returns a single value.
  - `n:` An integer specifying how many times func should be applied. Defaults to 1.

- **Output parameter:** The final value after `func` has been applied `n` times.
- **Input file format:**
  - The first line indicates the number of test cases.
  - Each subsequent test case will consist of:
    - A line representing the value. This can be an integer or a string.
    - A line with the name of the func function (as a string).
    - A line with the integer n. (If n is omitted, apply_n_times should use its default). If n is 0, the function should return the original value.
- **Example p2_input.txt content:**

```
3
5
double_it
3
hello
add_exclamation
2
10
square_it
1
```

Expected Output Example 1:

Given value = 5, func = double_it, n = 3
(i.e., double_it(double_it(double_it(5))))

Your function should produce the output: `40`
`{2*(2*(2*5)) = 40}`

Expected Output Example 2:

Given value = "hello", func = add_exclamation, n = 2
(i.e., add_exclamation(add_exclamation("hello")))

Your function should produce the output: `hello!!`

Notes and Restrictions:

- Ensure the function func is applied n times.
- You are not required to define the helper functions mentioned in the input (e.g., *double_it, square_it*). These will be provided during evaluation. We have provided some sample helper functions for convenience, however, during evaluation, there may be more/different helper functions.
- If `n` is `0`, func should not be applied at all, and the original value should be returned.
- The *value* and function *func* can be of generic types, as long as *func* is compatible with value.
- No external libraries needed.

Problem 3: Recursive Countdown Printer

Description:
Create a function named *countdown* that takes a single integer argument *n* and prints only the **odd numbers** from *n* down to 1, each on a separate line, using recursion. The function should not use loops (`for` or `while`). Instead, it should call itself recursively until the countdown reaches 1.

Function Definition:
You will define the function with the following signature:
`def countdown(n: int) -> None:`

Input parameter:
`n`: positive integer representing the starting number of the countdown.

Input File (`p3_input.txt`):

You can test your solution using input provided in the `p3_input.txt` file. You may add additional test cases to this file.

Input file format:

- The first line indicates the number of test cases.
- Each subsequent line contains a single integer `n`, representing the starting number for the countdown.

Example p3_input.txt content:

```
2
3
5
```

Expected Output Example:

```
3
1
5
3
1
```

Here 2 lines are printed for the first test case (n=3), and 3 lines are printed for the second test case (n=5), since only odd numbers are printed.

Notes and Restrictions:

- You must implement the solution **recursively** (no loops allowed).
- Assume `n` to be a positive integer, i.e. $n > 0$

Problem 4: Rotate String

Description:
Create a function named `rotate_string` that takes a string `s` and an integer `k`. The function should return a new string that is the result of rotating the original string `s` by `k` positions to the right.

- If `k` is positive, the rotation is to the right (e.g., the last character becomes the first).
- If `k` is negative, the rotation is to the left.
- The value of `k` can be larger than the length of the string.

Function Definition:

```
def rotate_string(s: str, k: int) -> str:
```

- **Parameters:**
  - `s`: The string to be rotated.
  - `k`: An integer representing the number of positions to rotate.
- **Returns:** The new, rotated string.

Input File (`p4_input.txt`):

- The first line is the number of test cases.
- Each test case consists of a single line containing the integer `k` and the string `s`, separated by a space.

Example `p4_input.txt`:

```
4
2 Python
-1 Hello
7 abc
0 xyz
```

Expected Output for Example:

For each line in the example input, your function will be called and should produce the following outputs:

```
onPyth
elloH
cab
xyz
```

The output is a result of a circular rotation of the string. The number of positions to rotate is determined by `k`.

- **Input:** `k=2, s="Python"` → **Output:** `"onPyth"`
  *The last **2** characters (`"on"`) are moved to the front.*
- **Input:** `k=-1, s="Hello"` → **Output:** `"elloH"`
  *A negative `k` means a left rotation. The first **1** character (`"H"`) is moved to the end.*
- **Input:** `k=7, s="abc"` → **Output:** `"cab"`
  *Rotation is circular. For a 3-character string, a rotation of 7 is equivalent to a rotation of 1. The last **1** character (`"c"`) is moved to the front.*

Notes:

- Your solution should correctly handle positive, negative, and zero values for `k`, as well as values of `k` whose absolute value is greater than the length of the string.
- The string length is >= 0.

Requested files

p1.py

```python
def filterByProperty(array: list, property: str, value: int) -> list:
    #write code here




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

def solution(tc):
    prop, val, arr = tc
    return filterByProperty(arr, prop, val)


def parse_object(line: str) -> dict:
    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):
        prop_val_line = lines[current_line_idx].split()
        prop = prop_val_line[0]
        val = int(prop_val_line[1])
        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((prop, val, objects_list))

    return input_tests


if __name__ == "__main__":
    Input = process_input('p1_input.txt')
    for prop, val, arr in Input:
        print(filterByProperty(arr, prop, val))

```

p1_input.txt

```
2
score 90
3
{"id": 1, "score": 90}
{"id": 2, "score": 85}
{"id": 3, "score": 90}
id 2
4
{"id": 1, "name": "Alice"}
{"id": 2, "name": "Bob"}
{"id": 3, "name": "Charlie"}
{"id": 4, "name": "David"}
```

p2.py

```python
def apply_n_times(value, func, n=1):
    #write code here




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

# Helper functions used for testing
# During evaluation, we may have more functions which will be passed as argument to your apply_n_times function.
# So, don't hardcode any usage of these functions.
def _double_it(x):
    return x * 2

def _add_ten(x):
    return x + 10

def _square_it(x):
    return x * x

def _add_exclamation(s):
    return s + "!"

_func_map = {
    "double_it": _double_it,
    "add_ten": _add_ten,
    "square_it": _square_it,
    "add_exclamation": _add_exclamation,
}

def solution(tc):
    inp_value, inp_func_name, inp_n = tc
    func_obj = _func_map[inp_func_name]
    return apply_n_times(inp_value, func_obj, inp_n)

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

        func_name = lines[current_line_idx]
        current_line_idx += 1

        n_str = lines[current_line_idx]
        current_line_idx += 1

        try:
            value = int(value_str)
        except ValueError:
            value = value_str

        n = int(n_str)

        input_tests.append((value, func_name, n))

    return input_tests

if __name__ == "__main__":
    Input = process_input('p2_input.txt')
    for tc in Input:
        solution(tc)

```

p2_input.txt

```
3
5
double_it
3
hello
add_exclamation
2
10
square_it
1
```

p3.py

```python
def countdown(n: int) -> None:
    # write your code here below




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

def solution(inp):
    return countdown(inp)

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

    for t in range(1, num_tests + 1):
        # Parse the test case (list of integers) for each line
        L = lines[t]
        input_tests.append(L)  # Append the list directly

    return input_tests

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

```

p3_input.txt

```
2
3
5
```

p4.py

```python
def rotate_string(s: str, k: int) -> str:
    # write code here




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

def solution(tc):
    s_val, k_val = tc
    return rotate_string(s_val, k_val)

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):
        parts = lines[i].split(' ', 1)
        k = int(parts[0])
        s = parts[1] if len(parts) > 1 else ""
        input_tests.append((s, k))

    return input_tests

if __name__ == "__main__":
    Input = process_input('p4_input.txt')
    for s_val, k_val in Input:
        print(rotate_string(s_val, k_val))
```

p4_input.txt

```
4
2 Python
-1 Hello
7 abc
0 abc
```

VPL