15-110 Fall 2019
Hw 04

Out: Friday 20th September, 2019 at 14:30 AST
Due: Thursday 26th September, 2019 at 17:30 AST

Introduction

In this homework you will practice with lists and tuples.

The total number of points available from the questions is 100.

In your `.zip` file (see general instructions below), you need to include only the file `hw04.py` with the python functions answering the questions in Sections 1 and 2. In the handout you have found a file `hw04.py` with the functions already defined but with an empty body (or partially filled). You have to complete the body of each function with the code required to answer to the questions.

General Instructions for Submitting the Assignments

Submissions are handled through Autolab, at https://autolab.andrew.cmu.edu/courses/15110q-f19

You are advised to create on your computer/account a folder named `110-hw`. For each new homework, you should create a new sub-folder named `01`, `02`, etc. where you can put the files related to the homework. In this way you work will be nicely organized and information and files will be easily accessible.
You can also create an equivalent struture for the *laboratories*, where in this case the root folder should be named `110-lab`.

When you are ready with the homework and want to submit your solutions, you need to go in the current homework folder (e.g., `01`), *select all files you will submit* (that can include both `.pdf` files with written answers to questions and python code files, `.py`) and compress them in one single `.zip` file.[1]

According to the OS you are using, you migh have different options for making the zip file. For instance, on Windows, after selection of the files, you should right-click and select `Send to: Compressed folder`, while on macOS, you can select `Compress` on the menu appearing from the right-click.

The compression action will produce a zip file containing the files to be handed in for the assignment. The file should be named `hwXX-handin.zip` (e.g., for this homework, the name of the file should be `hw04-handin.zip`). Then, open Autolab, find the page for this assignment, and submit your `hw04-handin.zip` file via the "Submit" link.

☛ The number of submissions is limited to 5. The last submission is the one that will be graded.

Style

Part of your grade on assignments are style points, that can be lost if your code is too disorganized, unreadable or unnecessarily complicated. To avoid loosing style points, please follow the guidelines at https://web2.qatar.cmu.edu/cs/15110/resources/style.pdf.

[1] The (single) zip file is needed, even when the files handed for the assignment consists of one individual file.

Note: Only the problems in Sections 1 and 2 will be graded. Problems in Section 3 are provided for further practice. Problems in Section 2 require concepts (list methods) that will be introduced in Sunday's lecture.

Problems that provide a partial solution code need to be completed in the `hw04.py` file, which is the only file that must be submitted on Autolab.

1 Problems not requiring the use of list methods

Problem 1.1: (10 points)

Implement the function `get_middle(l)` that takes a list as input and returns a list of two elements, where the first is the element in the middle of the list `l`, and the second is a string. This string is equal to `'E'` if the list has an even number of elements and is equal to `'O'` for an odd number of elements.

When the list has an even number of elements, the middle element to return is the last element of the first half of `l`.

If the list has no elements, the function returns `None`.

You must use an `if/elif/else` construct in the function.

For instance, `get_middle([1,2,3,4])` returns `[2, 'E']`, while `get_middle([1,2,3,4,5])` returns `[3, 'O']`.

Problem 1.2: (15 points)

Implement the function `info_tuple(first_name, last_name, major, gpa)` that receives the information of a student and returns an information tuple in the order: `info`, `last name`, `first name`, `major`, `gpa`.

The field `info` is a string that summarizes the information about the student in a formatted way. For instance, if the function is invoked as `info_tuple('John', 'smith', 'cs', 99.5)`, `info` is the following string: `'John␣Smith␣is␣majoring␣in␣CS,␣current␣GPA␣is␣99.5'`,

and the function returns the tuple:
`('John␣Smith␣is␣majoring␣in␣CS,␣current␣GPA␣is␣10.5', 'Smith', 'John', 'CS', 10.5)`

Ensure that first and last name are capitalized and that the major is uppercased (in both the string and the other elements of the tuple). Spaces and commas need to be included as in the example.

If any of the arguments `first_name`, `last_name` and `cs` is not a string, or `gpa` is not a number, then the function prints out `Wrong inputs!` and returns `None`.

Problem 1.3: (10 points)

Implement the function `join_ends(l1, l2)`, that takes two lists as input and returns a list that is the concatenation of `l1` with the reverse of `l2`.

For example, `join_ends( [1,2,3], ['a','b','c'] )` should return `[1,2,3,'c','b','a']`.

If the first and last element of the new list are the same, only this element is returned by the function and not the whole list.

The code of the function is partially written below. You need to add the missing parts.

```
def join_ends(l1, l2):
    L = []
    L += l1
    L +=
    if  L[0] == L[-1]:

    else:
        return
```

Problem 1.4: (15 points)

Implement the function `nested_lists(l, i, j, k)` that takes as input a list of lists, `l`, and three integers. The function returns the `k`-th character of the `j`-th element of the `i`-th list inside `l`, if such character exists and the `j`-th element is a string. Otherwise, the function returns `None`.

For instance, if the function is invoked as `nested_lists([ [1,2,3], [2,4], ['a', 'bcd', 'e'] ], 2, 1, 1)` the returned value is `'c'`. If `j`=0, `k`=0, `i`=2, the returned value is `'a'`. For `i` = 0 (and whatever values for `j` and `k`), the returned value is `None`.

The code of the function is partially written below. You need to add the missing parts.

```
def nested_lists(l, i, j, k):
    if len(l) >=
        if len(l[i])
            if type(l[i][j]) == str:
                if len(l[i][j])
                    return
    return None
```

Problem 1.5: (15 points)

Implement the function `swap(l, i, j)` that takes a list and two positions, and swaps the elements of these two positions. You can assume `i` and `j` are valid positions in `l`. The function returns the same list (i.e., a list with the same identifier) but with the swapped elements.

For example, `swap([1,2,3], 0, 2)` returns `[3,2,1]`.

Problem 1.6: (15 points)

Implement the function `to_thirds(l)` that takes as input a list `l` of numbers, splits it into three sublists of the same size, and returns a list with the sublists in the same order that they occur in `l`.

For example, `to_thirds([1,2,3,4,5,6])` should return the list `[[1,2],[3,4],[5,6]]`.

Moreover, the function checks which one among the three sublists has the first element with the largest value and prints out a message with the information. For instance, in the example above, the function prints out `'Sublist␣2␣has␣max␣value'` (the value is 5, while the first value for sublists 0 and 1 are 1 and 3, respectively).

If the list cannot be equally divided into three parts, the function returns `None` and doesn't make any printing out.

The code of the function is partially written below. You need to add the missing parts.

```
def to_thirds(l):
    if len(l)
        return None
    one_third =
    two_thirds = one_third * 2
    t = (l[0:one_third],             , l[two_thirds:])
    if t[0] > t[1]:
        if t[0] >
            print("Sublist␣0␣has␣max␣value")
        else:
            print(        )
    elif t[1] > t[2]:
        print("Sublist␣1␣has␣max␣value")
    else:
```

2 Problems requiring the use of list methods

Problem 2.1: (20 points)

Implement the function `slice_shift(l, pos_from, pos_to)` that takes as inputs a list `l` and two integers. The function *shifts* the two elements at positions `pos_from` and `pos_from + 1` to the position `pos_to` in the list. The new list, with the shifted items, is returned. This list must have a different identity compared to `l`.

For instance, `slice_shift([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], 4, 2)` returns
`[1, 2, 5, 6, 3, 4, 6, 8, 9, 10, 11]`

3 Ungraded problems (requiring list methods and functions)

Problem 3.1: (0 points)

Implement the function `list_sort(l, s)` that takes a list `l` as input and sort it according to the order specified by the input string `s`.

The function returns a new list, where:

1. the first element is the length of `l`;
2. the second element is a character, which is `'A'` if `s` is `'ascending'`, and `'B'` if `s` is `'descending'`;
3. the third element is a list with the same elements of `l`, sorted according to the value of `s`;
4. the fourth element is the original list `l`, unchanged.

Note that the operations inside the function must not change the original list `l`.

For instance, `list_sort( [2,3,1,5,4], 'ascending')` returns the list
`[5, 'A', [1, 2, 3, 4, 5], [2, 3, 1, 5, 4]]`

If the sorting string `s` is neither equal to `'ascending'` nor to `'descending'`, the function returns `None` and prints out `'Sorting␣flag␣not␣recognized'`

Problem 3.2: (0 points)

Implement the function `add_info(n)` that takes as input a list of numbers, `l`, and a number `n`. The function adds an information header list at the beginning of list `l`, and also removes the element `n` from `l`.

More precisely, the function, first, creates an information header list composed by four elements, where:

- the first element is the length of `l`;
- the second element is the minimum value among the numbers in the second half of `l`;
- the third element is the maximum value among the numbers in `l` except those at the first and the last positions;
- the fourth element is the number of times the number `n` occurs in the list `l`, if the number of occurrences is 0, a value -1 is written.

Once created, the function inserts the above four-element list at the beginning of the list `l` (i.e., the four-element list plays the role of an information header about `l`). Then, the first occurrence of `n` is removed from `l` (if `n` is present in `l`). Finally, the function returns the modified list `l`.

For instance, `add_info([1, 2, 3, 4, 5, 3, 7], 3)` returns `[[7, 3, 5, 2], 1, 2, 4, 5, 3, 7]`.
For `add_info([10, 2, 3, 4, 2, 3, 1], 8)`, returned list is `[[7, 1, 4, -1], 10, 2, 3, 4, 2, 3, 1]`.