Homework 4 - Strings and Lists

CS 1301 - Intro to Computing - Fall 2022

Important

- Due Date: **Tuesday, September 27th, 11:59 PM.**
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - TA Helpdesk
  - Email TA's or use class Piazza
  - How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
- Comment out or delete all function calls. Only import statements, global variables, and comments are okay to be outside of your functions.
- **Read the entire document before starting this assignment.**

The goal of this homework is for you to enhance your understanding of strings, and lists. The homework will consist of 5 functions for you to implement. You have been given HW04.py skeleton file to fill out. However, below you will find more detailed information to complete your assignment. Read it thoroughly before you begin.

Hidden Test Cases: In an effort to encourage debugging and writing robust code, we will be including hidden test cases on Gradescope for some functions. You will not be able to see the input or output to these cases. Below is an example output from a failed hidden test case:

```
Test failed: False is not true
```

Written by Cynthia Medlej (cmedlej3@gatech.edu) & Andrew Hsu (yhsu72@gatech.edu)

Decode String

Function Name: decodeString()
Parameters: encodedStr (`str`)
Returns: decodedList (`list`)
Description: You are participating in the scavenger hunt, and you were given a secret code to the location of the prize. To decode the secret code, you need to perform the following decoding operations: First, reverse the string. Next, get rid of any `'@'` character in the string. Then, change all uppercase characters to lowercase. Finally, turn the string into a list by splitting the string with the character `'#'`. If the encoded string is empty, just return an empty list.

```
>>> decodeString("a@b@#cde@!#!")
['!', '!edc', 'ba']
```

```
>>> decodeString("J@ckets#1301#!")
['!', '1031', 'stekcj']
```

Sunday Brunch Date

Function Name: goodBrunch()
Parameters: ashList(`list`), nickList (`list`)
Returns: brunchDecision(`list`)
Description: Ash and Nick want to have brunch somewhere fancy this weekend. However, they're having a hard time deciding where to go with all the options in Atlanta. Write a function that will return a list containing the places where Ash and Nick both want to go. If there is only one place they agree on, return the name of the place as a string.

If they can't agree on any place, return a string with the following format:

```
"Brunch at home it is!"
```

Note: The list of brunch places returned should be in **alphabetic order**.

```
>>> goodBrunch(["Le Bon Nosh", "Petit Chou"],
               ["Le Bon Nosh","Bread and Butterfly"])
"Le Bon Nosh"
```

```
>>> goodBrunch(["Butter Milk Kitchen", "Little Tart", "Petit Chou"],
                ["Little Tart","Atlanta Breakfast Club","Butter Milk Kitchen"])
["Butter Milk Kitchen","Little Tart"]
```

Dorm Map

Function Name: room_dist()
Parameters: dormMap (`list`), name1 (`str`), name2 (`str`)
Returns: distance (`int`)
Description: After moving into your dorm, you want to know how far down the hall you are from your friends. You are given a map of the hallway in a nested list format as follows:

```
[[room1a,room1b],[room2a,room2b],[room3a,room3b]...]
```

Each room, which is represented by a list, consists of two people, and the entire hall is represented by a list consisting of rooms in their physical order. Write a function that takes in the room map and names of 2 people, and return the distance between the rooms the two people stay in. For example, the distance between room1 and room3 will be 2. You may assume no one in the hall has the same name, and the given name will exist in the map.

Note: Distance returned must be positive.

```
>>> dormMap = [
  ["Avi","Rio"],["Tazio","Baker"],["Lesaiah","Andrew"],["Robin","Avery"]
]
>>> room_dist(dormMap,"Avi","Andrew")
2
```

```
>>> dormMap = [
  ["Avi","Rio"],["Tazio","Baker"],["Lesaiah","Andrew"],["Robin","Avery"]
]
>>> room_dist(dormMap,"Tazio","Baker")
0
```

Farmer's Market

Function Name: freshProduce()
Parameters: veggies (`list`), prices (`list`)
Returns: veggieList (`list`)
Description: Every weekend, you and your friends decide to cook dinner with fresh produce from the farmer's market. However, since you guys are college students on a budget, you can't really afford to splurge on vegetables. Write a function that takes in two lists: the first list will contain the name of the vegetables and the second list will contain their corresponding price in the same order as the veggies list. Your function should return a list that contains all the vegetables with prices below 4$ and the total cost of your purchase.

Note: If none of the vegetables match your budget, return an empty list.

```
>>> veggies = ["Potato","Onion","Shallot","Basil"]
>>> prices = [3.0,2.9,4.2,6]
>>> freshProduce(veggies,prices)
["Potato","Onion",5.9]
```

```
>>> veggies = ["Cucumber","Mushroom","Broccoli","Zucchini","Carrot"]
>>> prices = [1.2,5.5,3.7,2.5,3.9]
>>> freshProduce(veggies,prices)
["Cucumber","Broccoli","Zucchini","Carrot",11.3]
```

Find Roommate

Function Name: find_roommate()
Parameters: my_interests(`list`), candidates (`list`), candidate_interests(`list`)
Returns: match (`list`)
Description: You looking for roommates based on mutual hobbies. You are given a 3 lists: the first one (`my_interest`) contains all your hobbies, the second one (`candidates`) contains names of possible roommate, and last one (`candidate_interests`) contains a list of each candidates' interest in the same order as the candidate's list, which means that the interest of `candidates[0]` can be found at `candidate_interests[0]` and so on. Write a function that takes in these 3 lists and returns a list of candidates that has 2 or more mutual interests as you.

```
>>> my_interest = ["baseball","movie","e sports","basketball"]
>>> candidates = ["Josh","Chris","Tici"]
>>> candidate_interests = [["movie","basketball","cooking","dancing"],
                          ["baseball", "boxing","coding","trick-o-treating"],
                          ["baseball","movie","e sports"] ]

>>> find_roommate(my_interest, candidates, candidate_interests)
['Josh','Tici']
```

```
>>> my_interest = ["cooking","movie","reading"]
>>> candidates = ["Cynthia","Naomi","Fareeda"]
>>> candidate_interests = [["movie","dancing"],
                          ["coding","cooking"],
                          ["baseball","movie","online shopping"] ]

>>> find_roommate(my_interest, candidates, candidate_interests)
[]
```

Grading Rubric

Function	Points
decodeString()	20
goodBrunch()	20
room_dist()	20
freshProduce()	20
find_roommate()	20
Total	**100**

Provided

The `HW04.py` skeleton file has been provided to you. This is the file you will edit and implement. All instructions for what the functions should do are in this skeleton and this document.

Submission Process

For this homework, we will be using Gradescope for submissions and automatic grading. When you submit your `HW04.py` file to the appropriate assignment on Gradescope, the autograder will run automatically. The grade you see on Gradescope will be the grade you get, unless your grading TA sees signs of you trying to defeat the system in your code. You can re-submit this assignment an unlimited number of times until the deadline; just click the "Resubmit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW04.py` on Canvas.