Homework EC - Extra Credit

CS 1301 - Intro to Computing

Important

- Due Date: **Tuesday, November 24th, 11:59 PM**.
- This is an individual assignment. High-level collaboration is encouraged, **but your submission must be uniquely yours.**
- Resources:
  - 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 try your hand at some more challenging problems. The homework will consist of 4 functions for you to implement. You have been given `HWEC.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 Peter Han (phan38@gatech.edu), Brian Chuo (bchuo3@gatech.edu), Josh Tabb (jtabb6@gatech.edu) & Rajit Khanna (rkhanna23@gatech.edu)

Valid Parentheses

Function Name: validParentheses()
Parameters: a string with parentheses (`str`)
Returns: valid or not (`bool`)
Description: You're trying to finish your 1301 HW but you realize your parentheses '(' & ')' and square brackets '[' & ']' might not be matched up correctly. Write a function that will check if your code has the correct parentheses matched up together. All other characters (letters, digits, white space, quotations, etc.) should not affect your output.

```
>>> validParentheses("[('peter', 'han'), ('colin', 'tam')]")
True
```

```
>>> validParentheses("[3, 2, (1])")
False
```

Recursive Bubble Sort

Function Name: bubbleSort()
Parameters: a list (`list`), the list's length (`int`)
Returns: None (`NoneType`)
Description: Bubble sort is a simple sorting algorithm that will repeatedly swap adjacent elements if they are in the wrong order. More information can be found on how it exactly works here. A general pseudocode is provided in the link. Given the unsorted list and its length, write a function to sort the original list given without making a new list. You **MUST** implement this function recursively. If the list given is empty or contains 1 item, then there is nothing to sort, and thus, the list should not be altered.

Hint: You should be using loops in combination with recursion for this function.

```
>>> alist = [1,3,2,5]
>>> bubbleSort(alist, 4)
>>> print(alist)
[1,2,3,5]
```

```
>>> alist = [100,99,98,56,0]
>>> bubbleSort(alist, 5)
>>> print(alist)
[0,56,98,99,100]
```

Group Anagrams

Function Name: groupAnagrams()
Parameters: list of strings (`list`)
Returns: grouped anagrams (`dict`)
Description: An anagram is when two words have the same letters, but they can be in a different order. For example, *ab* and *ba* have the same letters, so we can say they are anagrams. Given a list of strings, write a function that returns the strings grouped by anagrams. We want you to think about how you can use a dictionary to see if two strings have the same letters.

You should be returning a dictionary where the keys are sorted letters and the values are a list of strings that are anagrams of those letters.

Let's explain the example from the first test case. The input list is

```
['ab', 'ba', 'bb', 'bba']
```

Our ouput should be of the form:

```
{
    'ab': ['ab', 'ba'],
    'bb': ['bb'],
    'abb': ['bba']
}
```

Since *ab* and *ba* are anagrams, we group them in a list. The key to this list in the dictionary is *ab*, which is *ab* and *ba* in alphabetical order. Since *bb* and *bba* don't have any anagrams in the list, they are grouped alone in lists. The keys to these lists in the dictionary are also *bb* and *bba* in alphabetical order.

Note: You can assume that you will not have to deal with any special characters, only lowercase letters. You can also assume you won't have to deal with any repeats in the input list.

Hint: It may help to use Python's `sorted()` function!

```
>>> strings = ['ab', 'ba', 'bb', 'bba']
>>> groupAnagrams(strings)
{'ab': ['ab', 'ba'], 'bb': ['bb'], 'abb': ['bba']}
```

```
>>> strings = ['rat', 'act', 'stressed', 'desserts', 'cat']
>>> groupAnagrams(strings)
{'art': ['rat'], 'act': ['act', 'cat'], 'deerssst': ['stressed', 'desserts']}
```

Winter Break Planner

Description: Winter Break is almost here, which means you want to start planning some virtual hang out sessions with your friends. To do this, you will be creating 2 classes: a **Friend** class and a **Planner** class. The implementation of these classes will be explained below.

Friend

Attributes:

- name (`str`): the name of the friend
- schedule (`dict`): A dictionary with days of the week as keys, and the activity planned for day as values. Note that there will only be keys for days where there is an activity planned. (e.g. `{'Monday': 'Homework', 'Friday': 'Among Us Game Night'}`)

Methods:

- __init__
    - initializes the following attributes from parameters:
        - name (`str`)
        - schedule (`dict`)
- addActivity
    - This method should take in a tuple, with a string of a day as the first element, and a string of an activity as the second element (E.g. `('Tuesday', 'Dentist')`).
    - If there is no activity planned on the given day in this `Friend` 's schedule, then add this day as a key into their schedule with the activity as its value.
    - If there is already an activity planned on that day, return `"Not possible"` .

Planner

Attributes:

- friendsList (`list`): list of `Friend` objects.

Methods:

- __init__
  - initializes the following attributes from parameters:
    - friendsList (`list`)
- freeTime
  - This method should not take in any parameters.
  - Based on the schedules of each `Friend` in `friendsList`, this method should return a list of the days of the week when every `Friend` in `friendsList` is free (i.e. no one has an activity planned for that day).
  - If there are no days where everyone is free, return `"No one is free"`.
- plans
  - This method should take in a day of the week as a string.
  - This method should return a dictionary where the keys are activities, and the values are a list of the names of `Friend`s in `friendsList` who are doing the same activity on the given day. See below for an example:

```
{
    'Among Us Game Night': ['Arushi', 'Arvin', 'Anthony'],
    'Spikeball': ['Jakob', 'Damian', 'Will', 'Caitlin']
}
```

Grading Rubric

Function	Points
validParentheses()	25
groupAnagrams()	25
bubbleSort()	25
Friend + Planner	25
Total	**100**

Provided

The `HWEC.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 `HWEC.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 `HWEC.py` on Canvas.