Homework 11

CS 1301 - Intro to Computing - Spring 2022

Important

- Due Date: **Tuesday, April 26th, 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 solve some of problems written by other students in the class for Lab02. The homework will consist of 5 functions for you to implement. You have been given `HW11.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 **the students of CS 1301 - Spring 2022!**

Puppy Finder

Function Name: puppyFinder()
Parameters: puppyCityDict (`dict`), cityDistanceDict (`dict`)
Returns: representation of puppy name, city and distance (`str`)
Description: You are looking to adopt a new puppy, and you want to find the closest one possible. Given a dictionary mapping puppy names (`str`) to cities (`str`) and a dictionary mapping cities (`str`) to distances (`int`), find the closest puppy. Return a string in the format:

```
"Your new puppy, {name}, is in {city} {miles} miles away."
```

Assume the distance will be at most 1000 miles and every city will appear exactly once in both dictionaries.

```
>>> puppyCityDict = {
        "Ralph": "Houston",
        "Bailey": "New York",
        "Buster": "San Francisco",
        "Cookie": "Charlotte"
}
>>> cityDistanceDict = {
        "Houston": 489,
        "New York": 673,
        "San Francisco": 893,
        "Charlotte": 237
}
>>> puppyFinder(puppyCityDict, cityDistanceDict)
'Your new puppy, Cookie, is in Charlotte 237 miles away.'
```

```
>>> puppyCityDict = {
        "Luna": "Johns Creek",
        "Teddy": "College Park"
}
>>> cityDistanceDict = {
        "Johns Creek": 38.1,
        "College Park": 9.7
}
>>> puppyFinder(puppyCityDict, cityDistanceDict)
'Your new puppy, Teddy, is in College Park 9.7 miles away.'
```

Double Odd Half Even

Function Name: doubleOddhalfEven()
Parameters: numberList (`list`)
Returns: changedList (`list`)
Description: Write a function called `doubleOddhalfEven()` that takes in a list of numbers as integers (numberList). This function doubles its odd entries and halves its even entries. Your new numbers should be returned as integers in a new list from smallest to largest (changedList). If the numberList given is empty, return an empty list. You must implement this function **recursively**.
Hint: Using `sorted()` on a list returns the sorted version of that list.

```
>>> doubleOddhalfEven([1,7,15,3,2,66])
[1, 2, 6, 14, 30, 33]
```

```
>>> doubleOddhalfEven([0,0,1,1,2,2,3,3])
[0, 0, 1, 1, 2, 2, 6, 6]
```

Dating App

Function Name: datingApp()
Parameters: profile1 (`list`), profile2 (`list`)
Returns: compatibility (`str`)
Description: You're a software developer whose latest job includes writing the code for a new dating app. You're given two lists that represent the user's profiles which will always be in the form `[name (str), age (int), interests (list)]`. Two people are considered compatible if they have an age gap of at most five years and at least three of the same interests. If they are compatible, you must find their compatibility using the ratio of their shared interests to the total number of interests (not counting duplicates) between both of them. Write a function called `datingApp()` that takes in two lists and returns a string in the following format if the pair is compatible:
Note: Round the compatibility to 1 decimal place.

```
"You're {compatibility}% compatible!"
```

Note: If the pair is not compatible return `Sorry, you're incompatible.`

```
>>> profile1=['Nora',20,['hiking','swimming','birdwatching','reading','cooking']]
>>> profile2=['Drew',21,['cooking','hiking','reading','running','researching']]
>>> datingApp(profile1, profile2)

"You're 42.9% compatible!"
```

```
>>> profile1=['Annie',24,['shopping','singing','baking','socializing','running']]
>>> profile2=['Sam',23,['surfing','running','swimming','exploring','reading']]
>>> datingApp(profile1, profile2)

"Sorry, you're incompatible."
```

Simplest Directions

Function Name: simplestDirections()
Parameters: directions(`str`)
Returns: simple direction (`str`)
Description: You are in New York City and Google Maps has crashed. You ask your friend for directions to the subway station and he gives you these directions in the format of a string. Your friend's directions are too confusing for you, so you decide to write a function to simplify these directions. In the string your friend gave you, `"<"` represents moving one step to the left, `">"` represents moving one step to the right, `"U"` represents moving one step up and `"D"` represents moving one step down. The function `simplestDirections()` should process the given string and return a string in the format:

```
"You have moved {num} blocks (up/down) and {num} blocks (left/right)."
```

Note: Movements in opposite directions cancel each other out.
Note: If there was no movement at all, return `'No movement.'`

```
>>> simplestDirections(">U<U>>DU<>U")

'You have moved 3 blocks up and 2 blocks right.'
```

```
>>> simplestDirections("><<>DUD")

'You have moved 1 blocks down and 1 blocks right.'
```

Song Mystery

Function Name: songMystery()
Parameters: codedSong (`str`), songNames (`list`)
Returns: newSong (`str`)
Description: Taylor Swift is about to release her new song, and she just released a clue for her fans to guess the name of the song. Given a name of a song (`str`) with the letter order randomized and a list of possible song names (`list`), return the correct name of the song (`str`). The letters will be a mix of upper and lower case letters, so make sure to account for case sensitivity. Return the correct name of the song in lowercase letters.

If you cannot find the song name in the list of possible song names given, return `'I need more clues :('`

```
>>> codedSong = "Ainga gineB"
>>> songNames = ["willow", "evermore", "Begin Again", "Lover", "gold rush"]
>>>songMystery(codedSong,songNames)

'begin again'
```

```
>>> codedSong = "byubuuv"
>>> songNames = ["willow", "evermore", "Begin Again", "Lover", "gold rush"]
>>> songMystery(codedSong,songNames)

'I need more clues :('
```

Grading Rubric

Function	Points
puppyFinder()	20
doubleOddhalfEven()	20
datingApp()	20
simplestDirections()	20
songMystery()	20
Total	**100**

Provided

The `HW11.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 `HW11.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 `HW11.py` on Canvas.