Homework 6 - Dictionaries and Try/Except

CS 1301 - Intro to Computing - Fall 2022

Important

- Due Date: **Tuesday, October 11th, 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 dictionaries and Try/Except blocks. The homework will consist of 5 functions for you to implement. You have been given the `HW06.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 Jane Crowley (jcrowley38@gatech.edu) & Farah Alkadri (falkadri3@gatech.edu)

Dining Hall Dinner

Function Name: dinnerTime()
Parameters: meal (`str`), diningHalls (`dict`)
Returns: optionsList (`list`)
Description: It's time for dinner and you're trying to decide where to go. Given a specific meal you want to eat and a dictionary where the keys are potential dining locations and the values are tuples of meal options, write a function that returns a list of the dining halls that have the meal you want. If no dining location has the meal you want, return the string `"Eat at home!"`

```
>>> diningHalls = {
  "Willage" : ("Mystery Meat", "Taco Salad", "Quinoa Casserole"),
  "Brittain" : ("Veggie Burger", "Cheeto Sushi", "Cereal"),
  "Nave" : ("Cereal",)
}
>>> meal = "Cereal"
>>> dinnerTime(meal, diningHalls)
[Brittain, Nave]
```

```
>>> diningHalls = {
  "Whistle Bistro" : ("Pickles", "Sandwich", "Burgers"),
  "Cluck N Mooh" : ("Milkshakes", "Chicken", "Burgers"),
  "Chick Fil A" : ("Chicken", "Milkshakes")
}
>>> meal = "Spaghetti"
>>> dinnerTime(meal, diningHalls)
"Eat at home!"
```

Actor Rating

Function Name: rateActor()
Parameters: skillScore (`list`), fanScore (`list`)
Returns: totalScores (`dict`)
Description: Lots of new movies have been coming out and it's up to you to rate all the actors. Given two lists, one a list of tuples with an actor and their acting skill score (`int` or `float`) and one a list of tuples with an actor and their fan score, write a function that returns a dictionary where the key is the actor name and the value is their total score. Some of the scores have been altered by haters, so using try/except, if the score is of the wrong type (not `int` or `float`), add 3 points to an actor's score.

```
>>> skillScore = [("Harry Styles", 1), ("Florence Pugh", 5), ("Chris Pine", 4),
                  ("Olivia Wilde", "%%%%")]
>>> fanScore = [("Harry Styles", 5), ("Florence Pugh", 5), ("Chris Pine", None),
                ("Olivia Wilde", 2)]
>>> rateActor(skillScore, fanScore)
{"Harry Styles" : 6, "Florence Pugh" : 10, "Chris Pine" : 7, "Olivia Wilde" : 5}
```

```
>>> skillScore = [("Tom Cruise", 3), ("Miles Teller", [5]),
                  ("Glen Powell", 3), ("Jennifer Connelly", {2:2})]
>>> fanScore = [("Tom Cruise", 4), ("Miles Teller", 5),
                ("Glen Powell", (2,)), ("Jennifer Connelly", 2)]
>>> rateActor(skillScore, fanScore)
{"Tom Cruise" : 7, "Miles Teller" : 8, "Glen Powell" : 6, "Jennifer Connelly" : 5}
```

Stock Market

Function Name: theRealDeal()
Parameters: stockDict (`dict`), goodStock (`str`)
Returns: richList (`list`)
Description: You have been investing your money in many stocks, but now it's time you analyze your investements and find out which stocks will make you rich for real! Write a function that takes two parameters: a stock dictionary and a string representing the highest selling stock. The stock dictionary (`stockDict`) contains the names of the stocks(`str`) you invested in as the keys, and a tuple containing the price(`float`), quantity of stocks(`int`), and percentage rate of change(`float`) of the specific stock. The function should return a list containing the total earning of the good stock and the names of the stocks with earnings greater than $7000. If you haven't invested in this goodStock, your function should return a list containing the string, `"You haven't invested in this one yet!"` and the names of the stocks with earnings greater than $7000.

Note: The names of the stocks with earning above $7000 should be added in **alphabetic order**.
Hint: You can calculate the earnings from a stock by using the formula: `earnings = price * quantity * rate of change of the stock`

```
>>> stockDict = {"LLY":(323.86,10,3.98),"ENPH":(277.47,50,-0.72),
                 "NBIX":(106.21,100,2.62)}
>>> goodStock = "OXY"
>>> theRealDeal(stockDict, goodStock)
["You haven't invested in this one yet!", 'LLY', 'NBIX']
```

```
>>> stockDict = {"Apple":(138.20, 1000,3),"GameStop":(25.13, 100, 1.3),
                 "Google":(95.65, 10, 1.82),"Amazon":(113,75,1.57)}
>>> goodStock = "Apple"
>>> theRealDeal(stockDict, goodStock)
[414600.0, 'Amazon', 'Apple']
```

Delta

Function Name: flightsInfo()
Parameters: flightsDict(`dict`)
Returns: fixedflightsDict(`dict`)
Description: You're interning for Delta this semester, and you were tasked to fix some issues with their flight information. Write a function that takes a flight dictionary (`flightsDict`) and returns a new dictionary with all of the correct flight information.

The flight dictionary(`flightDict`) is a dictionary with the flight names as the keys and a dictionary containing the flight details as the values. The flight dictionary will always be in the format:

```
flightsDict = {'flight1': {'time': 'time1',
                           'weather': 'weather1',
                           'passengers': [('Name', 'Seat', 'Zone'),('Name', 'Seat'),
                                          ('Name', 'Seat')],
                          },
               'flight2': {'time': 'time2',
                           'weather': 'weather2',
                           'passengers': [('Name', 'Seat'), ('Name', 'Seat')],
                          }
                    .
                    .
                    .
               }
```

For each flight in the flight dictionary, there is a key called passengers that contains a list of tuples as its values. Each tuple is supposed to contain the passenger's name, their seat number and their zone, but you'll notice that some passengers are missing zone assignments. Your task is to return a fixedflightsDict(`dict`) that will have the missing zones added to the passengers' value. There are three boarding zones: zone `"A"` for passengers with 0 - 5 letter names, zone `"B"` for pasengers with 6 letter names, and `"Premium"` passengers with 7 or more letters in their name.

```
>>> flightsDict = {"flight1":
                   {
                    "time":"6:00am",
                    "weather":"windy",
                    "passengers": [("Farah","20A","A"), ("Jane","4A"),
                                   ("Naomi", "3B"),("Fareeda", "1B","Premium'")]
                   },
                   "flight2":
                   {
                     "time":"3:00 pm",
```

```
                    "weather":"sunny",
                    "passengers": [("Nelson","2C"), ("Ramya","4A")]
                   }
                  }
>>> flightsInfo(flightsDict)
{
  "flight1":{
  "time":"6:00 am",
  "weather":"windy",
  "passengers": [("Farah","20A","A"), ("Jane","4A","A"),
                 ("Naomi","3B","A"), ("Fareeda", "1B","Premium")]
  },
 "flight2":{
   "time":"3:00 pm",
   "weather":"sunny",
   "passengers": [("Nelson","2C","B"), ("Ramya","4A", "A")]
 }
}
```

```
>>> flightsDict = {"flight1":
                   {
                    "time":"6:00am",
                    "weather":"windy",
                    "passengers": [("Anastasiya","2B"), ("Chris","4A"),
                                   ("Naomi", "4B", "A"),("Andrew", "10A", "B")]
                   }
               }
>>> flightsInfo(flightsDict)
{"flight1":
 {
   "time":"6:00am",
   "weather":"windy",
   "passengers": [("Anastasiya","2B","Premium"), ("Chris","4A", "A"),
                  ("Naomi", "4B", "A"),("Andrew", "10A","B")]
 }
}
```

Fast Food Revisited

Function Name: fastFood()
Parameters: friendDict (`dict`), menu (`dict`)
Returns: friendsOwed (`list`)
Description: Remember the fastFood() homework problem in HW05? Now you have learned the concept of Dictionary, you will see how powerful this data structure is by revisiting a similar problem. You are given a dictionary containing your friends and the food they bought.

```
{
  "friend1": ["food1", "food2", ...],
  "friend2": ["food3", "food4", ...],
  .
  .
  .
}
```

You are also given a dictionary mapping the food to its price (`float`). Such as

```
{
  "food1": price1,
  "food2": price2,
  .
  .
  .
}
```

Similar to the homework last week, write a function fastFood() that returns a list of tuple containing the (name of friend, total amount they owe) in alphebetical order.

```
>>> friendDict = {
  "Chris": ["coke", "cookie"],
  "Andrew": ["fries", "burger"],
  "Naomi": ["burger"]
}
>>> menu = {
  "coke": 1.0,
  "fries": 3.0,
  "burger": 10.0,
  "cookie": 2.0,
}
```

```
>>> fastFood(friendDict,menu)
[("Andrew", 13.0),("Chris",3.0),("Naomi", 10.0)]
```

```
>>> friendDict = {
  "Paige": ["burger", "burger","burger"],
  "Aryan": ["coke", "fries"],
  "Fareeda": ["coke", "fries"]
}
>>> menu = {
  "coke": 1.0,
  "fries": 3.0,
  "burger": 10.0,
  "cookie": 2.0,
}
>>> fastFood(friendDict,menu)
[("Aryan", 4.0),("Fareeda",4.0),("Paige", 30.0)]
```

Grading Rubric

Function	Points
dinnerTime()	20
rateActor()	20
theRealDeal()	20
flightsInfo()	20
fastFood()	20
Total	**100**

Provided

The `HW06.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 `HW06.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 `HW06.py` on Canvas.