Homework 5 - Tuples and Modules

CS 1301 - Intro to Computing - Fall 2024

Important

- Due Date: **Thursday, October 3rd, 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 Ed Discussion
  - 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.**

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: Collin Kelly, Nick Byrd, & Priya Annapureddy

Helpful Information to Know

Modules

Modules allow you to use code from another source. This could include other code that you've written in a different Python file, or external code written by other programmers. To use a module, you must **import** it into your file first. There are multiple ways to do this. Let's look at some ways to import and use the `pi` constant and `sqrt` function from Python's built-in **math** module.

Note: By convention, `import` statements should be at the top of your file, **not in a function.**

- Importing the module itself:

```
import math
val = math.pi * math.sqrt(4)
```

- Importing specific item(s) from a module:

```
from math import pi, sqrt
val = pi * sqrt(4)
```

- Importing a module with a different name:

```
import math as m
val = m.pi * m.sqrt(4)
```

Cook Off!

Function Name: chiliChampion()
Parameters: nameList (`list`), scoreList (`list`)
Returns: winner (`str`)
Description: You got lucky enough to be a judge for the town's annual chili cook off! Scores are tight, and the you need a way to determine which chili won.

Write a function called `chiliChampion()` that takes in two parameters: a list of the names of the contestants, and a list of lists containing scores for each contestant. Each chef's name in `nameList` will be at the same index of the list of scores for their chili in `scoreList`. For each contestant, find the average score rounded to two decimal places and **print** out a string of the form `"{contestantName}: {avgScore}"`. Once you have found the average score for each chef, **return** a string of the form `"{winningChef} wins with a whopping {points} points!"`.

Note: You can assume that there will be no ties and that the lengths of `nameList` and `scoreList` will be the same.

```
>>> nameList = ["Chris", "Collin", "Priya"]
>>> scoreList = [[8.9, 8.2, 7.9], [7.3, 9.4, 8.8], [8.5, 8.7, 9.2]]
>>> chiliChampion(nameList, scoreList)
"Chris: 8.33"
"Collin: 8.5"
"Priya: 8.8"
'Priya wins with a whopping 8.8 points!'
```

```
>>> nameList = ["Mikel", "Declan", "Kai"]
>>> scoreList = [[9.2, 7.4, 8.9], [6.4, 8.1, 6.8], [9.8, 9.2, 9.0]]
>>> chiliChampion(nameList, scoreList)
"Mikel: 8.5"
"Declan: 7.1"
"Kai: 9.33"
'Kai wins with a whopping 9.33 points!'
```

Autumn Activities

Function Name: autumnBucketList()
Parameters: allActivities (`list`)
Returns: chosenActivities (`list`)
Description: Since it's now officially fall, you and your friends want to plan some autumn activities!

You have compiled a list of fall-themed activities which contain tuples with an activity and how many votes it has in the format `(activity, votes)`. If all activities have 0 votes, **print** the string `"Is it summertime yet?"`. Otherwise, **print** out details about the highest rated activity as string in the format `"With {highest_votes} votes, {favorite_activity} wins!"`. Then, **return** the list of activities sorted by highest rating, with all the activities with 0 votes removed.

```
>>> allActivities = [('Haunted House', 6),
                     ('Pumpkin Patch', 9), ('Apple Picking', 2), ('Corn Maze', 1)]
>>> autumnBucketList(allActivities)
"With 9 votes, Pumpkin Patch wins!"
['Pumpkin Patch', 'Haunted House', 'Apple Picking', 'Corn Maze']
```

```
>>> allActivities = [('Baking', 8), ('Reading', 4),
                     ('Raking Leaves', 0), ('Trick O Treating', 0)]
>>> autumnBucketList(allActivities)
"With 8 votes, Baking wins!"
['Baking', 'Reading']
```

Apple Picking

Function Name: applePicker()
Parameters: schedule (`list`)
Returns: dates (`list`)
Description: Since starting college, you are so busy you now have to plan some of your events years in advance. You would like to go apple-picking, but there are a few hang ups.

First, you cannot pick apples on a leap year. Second, you can only pick apples on the weekend (Saturday or Sunday) because you have class every day in the week. Thus, given a list of dates in the format `'MM-DD-YYYY'` create and **return** a new list of dates that are suitable to go apple picking in the new format `["{Month} {Day}, {Year}", ...]`. If there are no days you can go apple picking, **return** the string `"Sorry, we cannot pick any apples :("`. Assume all dates will have two digits for the day and month (i.e. January 1 2024 is "01-01-2024").

NOTES:

- Reading the documentation for the calendar module will be very helpful! The `weekday()` and `isleap()` functions are especially useful (https://docs.python.org/3/library/calendar.html#module-calendar)
- There is a *data attribute* in the calendar module that can convert numbers to month names!

```
>>> applePicker(["01-16-2003","01-09-2004","10-02-2022"])
['October 2, 2022']
```

```
>>> applePicker(["09-08-2024"])
"Sorry, we cannot pick any apples :("
```

Hayride Havoc

Function Name: hayrideHero()
Parameters: coordinateList (`list`)
Returns: distanceStr (`str`)
Description: You recently got a job driving the hayride at this year's fall festival! Unfortunately, guests keep asking about the length of this ride, but you're unsure about the exact distance.

Write a function that takes in a list of tuples that give the name of the hayride stop as well as its co-ordinates, formatted as `(hayrideStop, x_coordinate, y_coordinate)`. Starting at the beginning of the ride, location `(0,0)`, calculate the straightline distance between *each* stop and sum to find the total distance traveled. Then, **return** a string of the form `"This hayride is {totalDistance} miles long!"`, with the total distance of this year's ride **rounded** to two decimal places.

Notes:

- The distance between two points (x_1, y_1) and (x_2, y_2) is given by $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1))^2}$, where the $\sqrt{}$ symbol represents the taking the square root of the entire part to follow. Last week, you found the square root without using the math module. Now, you should import the math module and use the square root function!
- The ride begins at the coordinates `(0,0)`.

```
>>> coordinateList = [("Pumpkin Patch", 0.2, 0.4),
                      ("Corn Maze", 0.5, 0.7), ("Scarecrow Field", 0.6, 0.9)]
>>> hayrideHero(coordinateList)
"This hayride is 1.1 miles long!"
```

1.1 miles = x miles from (0, 0) to (0.2, 0.4) + y miles from (0.2, 0.4) to (0.5, 0.7) + z miles from (0.5, 0.7) to (0.6, 0.9)

```
>>> coordinateList = [(("Apple Orchard"), 0.4, 0.7),
                      ("Harry's Barn", 0.4, 0.9), ("Pie Peak", 0.2, 0.1)]
>>> hayrideHero(coordinateList)
"This hayride is 1.83 miles long!"
```

Leaves Jumping Contest

Function Name: jumpKing()
Parameters: personList (`list`)
Returns: winnerAndLoser (`str`)
Description: Since you did so well with the chili competition, you've been asked to judge the first ever annual CS1301 TA Leaf Jumping Contest!

The TAs have given you a module called **leafTally.py** that uses the TAs' special scoring criteria. Download this module and **store it in the same folder as your `HW05.py` file.** Do not submit leafTally.py to Gradescope.

Write a function that takes in a list of names and results in the format `[[name1, result1],[name2, result2],...]`. The results themselves are nested lists of colors of leaves the TA jumped on, in the format `["color1", "color2", "color3",...]`. Using the function `leafTally()` in the `leafTally.py` module, find the TAs with the best and the worst score. If there aren't at least two TAs in the list, return the string `"Please enter 2 TAs!"`. Otherwise, return the string `"{TA_winner} wins with {bestPoints} points! {TA_loser} lost with only {worstPoints} points..."`

Notes:

- Assume there will never be a tie.
- The `leafTally()` function in the `leafTally.py` module takes in a string with only the first letter of each color as an input (for example, `"roy"` for the list `["red", "orange", "yellow"]`). Open the module and read `leafTally()` to get an idea of how the function works.

```
>>> personList = [["Collin", ["red", "orange", "yellow"]],
                  ["Priya", ["orange", "yellow"]], ["Chris",["orange", "orange"]]]
>>> jumpKing(personList)
"Collin wins with 85 points! Chris lost with only 20 points..."
```

```
>>> jumpKing([["Collin", ["red", "orange", "yellow"]]])
"Please enter 2 TAs!"
```

Grading Rubric

Function	Points
chiliChampion()	20
autumnBucketList()	20
applePicker()	20
hayrideHero()	20
jumpKing()	20
Total	**100**

Provided

The `HW05.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 `HW05.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 `HW05.py` on Canvas.

Do not submit leafTally.py to Gradescope.