Homework 4 - Strings and Lists

CS 1301 - Intro to Computing - Fall 2024

Important

- Due Date: **Thursday, September 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 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: Saleh Alsedrah, Kirito Machida, Danny Nasibu, & Priya Annapureddy

Movie Munchies

Function Name: snackMaker()
Parameters: orders (`list`), snacks (`list`), drinks (`list`)
Returns: moviegoers (`list`)
Description: You work at the movie theater snack station, and with everyone coming for the Interstellar re-release, you've run out of many items!

Write a function that takes three lists: a orders list containing lists in the format `[name, snack, drink]`, a list of snacks you have, and a list of drinks you have. **Return** the people whose orders you have the snacks **and** drinks to make as a list, **sorted** alphabetically.

```
>>> orders = [["Sila", "Olive Oil Popcorn", "Cola Icee"],
              ["Danny", "Popcorn", "Fanta"]]
>>> snacks = ["Olive Oil Popcorn", "Pretzel", "Hot Dog"]
>>> drinks = ["Cola Icee", "Sprite", "Lemonade"]
snackMaker(orders, snacks, drinks)
['Sila']
```

```
>>> orders = [["Saleh", "Skittles", "Coke"], ["Kirito", "Hot Dog", "Icee"]]
>>> snacks = ["Junior Mints", "Skittles", "Hot Dog", "Popcorn"]
>>> drinks = ["Root Beer", "Icee", "Coke", "Fanta", "Water"]
>>> snackMaker(orders, snacks, drinks)
['Kirito', 'Saleh']
```

(P.S. These are actually your TAs favorite movie snacks!)

The Gravity Equation

Function Name: gravityEquation()
Parameters: calculations (`str`)
Returns: correctResults (`list`)
Description: As a new NASA hire, you have been tasked with helping Murph, a brilliant NASA scientist, solve the gravity equation.

You are given a parameter, calculations (`str`), which contains multiple "calculations" separated by underscores (`_`). Each calculation is mix of digits and letters. Write a function that analyzes each calculation by counting how many digits and letters it contains. If the number of digits and letters in the calculation are equal, then that calculation is considered valid. For each valid calculation, extract and store only its digits. **Return** a **sorted** list of these **integers** in ascending order.

```
>>> calculations = "A1DA30W1_B63O23RE12L_TH19W9071G_4DF0B4_43M11AE0CR_K9SF80F6VA65"
>>> gravityEquation(calculations)
[404, 1301, 43110, 980665]
```

```
>>> calculations = "31PR4I1Y2A_6S2A83LE1H_6DA62N6N0NY1_KI602RI21T4O_2B36UZ71Z"
>>> gravityEquation(calculations)
[31412, 62831, 602214, 662601]
```

Hidden References

Function Name: findReference()
Parameters: message (`str`)
Returns: decryptedMessage (`str`)
Description: You are going to see the 10th year anniversary re-release of Interstellar and your friends keep making references to the movie but... you haven't seen it yet!

Write a function that takes in a parameter, message (`str`), where each word is separated by spaces. Parse the string to create two separate phrases: one containing all the lowercase words (using all the lowercase letters and spaces within the message) and another containing all the uppercase words (using all the uppercase letters and spaces within the message). To decrypt the message, **reverse** the phrases and **remove** all characters that are not letters or spaces. **Return** the final decrypted string, where the lowercase phrase is placed before the uppercase phrase. Be sure to **remove** any spaces at the end of your decrypted message before returning it.

```
>>> findReference(" Do7!L.l'R0l2(Oe)hW1")
"hello WORLD"
```

```
>>> message = " EtR9.iE4H hEI#trDae OnT1[o?' nTrNAE^o0Mb sREVaENw dSn3_=i=A8k}namW5"
>>> findReference(message)
"mankind was born on earth it WAS NEVER MEANT TO DIE HERE"
```

Wormhole Jumping

Function Name: getNextCoordinate()
Parameters: wormholeCoordinates (`list`), currentLocation (`list`)
Returns: nextCoordinate (`string`)
Description: In the movie Interstellar, astronauts embark on a mission through a mysterious wormhole near Saturn to find a new habitable planet for humanity, and you've recently joined the team.

Write a function that takes in two parameters: wormhole coordinates, a list of lists formatted as `[x, y, z, locationName]`, and your current coordinates, a list formatted as `[x, y, z]`. Your function should find the coordinate for the closest wormhole relative to your current location and the distance it takes to get there. If the lowest distance is less than or equal to 50, **return** the string `"The next coordinate is {next_coordinate}, we can go!"`, where `next_coordinate` is a list in the format `[x, y, z]` of the closest wormhole. Otherwise, **return** the string `"Oops..., not enough fuel to make it to {locationName}!"`. If there is a tie between multiple locations, you should use name of the one that appeared last in `wormholeCoordinates`.

NOTE:

- The distance between points (x_1, y_1) and (x_2, y_2) is given by $\sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2)}$, where the $\sqrt{}$ symbol represents the taking the square root of the entire part to follow.

```
>>> wormholeCoordinates = [[15, 25, 35, "Alpha Centauri"], [12, 22, 32, "Proxima"],
                           [8, 18, 20, "Sirius"], [40, 50, 10, "Vega"]]
>>> currentLocation = [10, 20, 30]
>>> getNextCoordinate(wormholeCoordinates, currentLocation)
"The next coordinate is [12, 22, 32], we can go!"
```

```
>>> wormholeCoordinates = [[20, 30, 400, "Betelgeuse"], [10, 150, 25, "Rigel"],
                           [500, 10, 15, "Antares"], [35, 405, 55, "Deneb"]]
>>> currentLocation = [17, 5, 34]
>>> getNextCoordinate(wormholeCoordinates, currentLocation)
"Oops..., not enough fuel to make it to Rigel!"
```

Which Planet?

Function Name: timeDilationCalculator()
Parameters: planetNames (`list`), planetDetails (`list`)
Returns: dilationCalculations (`list`)
Description: Cooper, a trained NASA pilot, and the other astronauts are deciding which planet to investigate, but they must do some calculations first.

You are tasked with creating a time dilation calculator that takes two lists: names of planets, and lists of their corresponding details in the format `[orbit_speed, gravitiational_force]`. If a planet has a gravitational force larger than 20, then you must **print** the string `"{planetName}'s gravitational force is too strong."` and **omit** it from further calculations and lists. Then, you must **print** out the name of the planet with the lowest time dilation in a string formatted as `"The lowest dilation is {planet_name}.`. Finally, **return** a **sorted** list of strings containing in the format of `"{planet}: {dilation}"`, with all calculations rounded to two decimal places.

Note: To calculate a planet's time dilation, you must divide its orbiting speed by the cube of its gravitational force.

```
>>> planetNames = ["Kepler-22b", "LHS 1140 b", "HD 209458b"]
>>> planetDetails = [[101266.2738, 19.3], [90091.13, 21.6], [68154.88, 8.13]]
>>> timeDilationCalculator(planetNames, planetDetails)
LHS 1140 b's gravitational force is too strong.
The lowest dilation is Kepler-22b.
['HD 209458b: 126.83', 'Kepler-22b: 14.09']
```

```
>>> planetNames = ["Proxima Centauri b", "WASP-12b", "TRAPPIST-1e", "GJ 1214b"]
>>> planetDetails = [[72629.07, 22.28], [85233.89, 8.27],
                     [22511.88, 15.25], [66244.74, 12.7]]
>>> timeDilationCalculator(planetNames, planetDetails)
Proxima Centauri b's gravitational force is too strong.
The lowest dilation is TRAPPIST-1e.
['GJ 1214b: 32.34', 'TRAPPIST-1e: 6.35', 'WASP-12b: 150.69']
```

Grading Rubric

Function	Points
snackMaker()	20
gravityEquation()	20
findReference()	20
getNextCoordinate()	20
timeDilationCalculator()	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.