Homework 4 - Strings, Indexing, & Lists

CS 1301 - Intro to Computing - Spring 2020

Important

- Due Date: **Tuesday, February 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.**

Responsible Coding: Some functions in this assignment will have a 'Responsible Coding' badge next to them. These functions are written to help you think about how you can use your skills as a programmer to analyze problems related to *sustainability*, such as ethics, health, security, and the environment.

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 Jasmine Chacko (jasminevchacko@gmail.com) & Rajit Khanna (rkhanna23@gatech.edu)

Message Encoder

Function Name: messageEncoder
Parameters: message (`string`)
Returns: encoded_message (`string`)
Description: Write a function that takes in a message and outputs this message with the words at every other position beginning with the first one in upper case, and every other word in between in lower case. That is, the first word should be upper case, the second word lower case, the third word should be upper case, and so on. If the string passed in is the empty string, the function should return **None**.

Hint: The string methods **.lower()** and **.upper()** might be useful. **lower()** ignores all punctuation to return a string with every letter in lowercase. **upper()** likewise ignores punctuation to return a string's uppercase representation.

```
>>> message = "I woke up, couple mil on my plate"
>>> print(messageEncoder(message))
I woke UP, couple MIL on MY plate
```

```
>>> message = "I just went gave my mama a hundred"
>>> print(messageEncoder(message))
I just WENT gave MY mama A hundred
```

Course Creator

Function Name: courseCreator
Parameters: courseList (`list`)
Returns: newCourse (`string`)
Description: Write a function that takes in a list of courses and returns a new course name. This course name should have the same department abbreviation as the courses in the course list (i.e. if the course list contains computer science courses the new course should start with "CS"). This new course's course number should be the average of all the course numbers of the courses in the course list (e.g. if the list passed in is `["CS1301", "CS1331"]` the ouptut should be `CS1316`). You may assume that all the courses in the course list will be in the same department. The average of all the course numbers will always be an integer (how do we make sure this is the case?). If the empty list is passed in, the function should return **None**.

Note: The length of the list can vary!

```
>>> courseList = ["CS1301", "CS1331", "CS1332", "CS2316", "CS4400"]
>>> print(courseCreator(courseList))
CS2136
```

```
>>> courseList = ["MATH1551", "MATH1552", "MATH1554", "MATH2551", "MATH2552"]
>>> print(courseCreator(courseList))
MATH1952
```

Compound Words

Function Name: compoundWords
Parameters: wordsList (`list`), num (`int`)
Returns: matchedWords (`list`)
Description: Write a function that takes in a list of strings and an integer. The function should create a list of strings of length the integer passed in (`num`) by concatenating the strings in the original list together. If the empty list is passed in, the function should return **None**.

Note: The function should ignore duplicates. In the first example below, the words firehose and hosefire are both of length 8. However, the function only outputs firehose because these two strings are considered equivalent. **In general, the function should only concatenate a string with strings that occur later (at higher indices) in the list.**

```
>>> wordsList = ["fire", "god", "speed", "trucks", "hose"]
>>> print(compoundWords(wordsList, 8))
['firehose', 'godspeed']
```

```
>>> wordsList = ["school", "time", "books", "work", "food"]
>>> print(compoundWords(wordsList, 10))
['schooltime', 'schoolwork', 'schoolfood']
```

Note Taker Responsible Computing

Function Name: noteTaker
Parameters: notes (`string`)
Returns: newNotes (`string`)
Description: The Office of Disability Services here at Georgia Tech offers accommodations for students with disabilities, including providing note-takers for students. Students in a class can volunteer to upload their notes to a portal, so that students who need them can access them. A note-taker should make an effort to ensure that their notes are clear, so that anyone reading them can understand them. Write a function that takes out extraneous punctuation within a group of sentences. The function should remove any and all punctuation within a sentence, but not its end punctuation.

Note: You only need to account for these characters: `. , ? ! @ :`

Hint: The **.isupper()** function can help you determine if a character is a capital letter! Every sentence after the first one will begin with a space and a capital letter.

```
>>> notes = 'Li!sts are a co.ol datat,ype! They can hold a@ lot of infor:!matio
n.'
>>> noteTaker(notes)
Lists are a cool datatype! They can hold a lot of information.
```

```
>>> notes = 'CS1301,,,-- wh:at a c@!ool c?lass!'
>>> noteTaker(notes)
CS1301-- what a cool class!
```

Food Deserts Responsible Computing

Function Name: foodDesertLocator
Parameters: cities (`list`)
Returns: city_list (`list`)
Description: A food desert is an area that has limited access to affordable and nutritious food; in other words, there aren't grocery stores within a certain mile radius to provide food to the people who live there. Write a function that takes in a list of cities and the number of people who are in that city without access to fresh food. Your function should first calculate the average number of people affected and then return a list of the city names with populations above that average.

Note: The length of the list can vary! The list will be structured as follows: the name of the city, followed by that city's population, and so on. You can assume that every city will be followed by its population.

```
>>> cities = ['San Francisco', 125000, 'Chicago', 500000, 'Atlanta', 2000000]
>>> foodDesertLocator(cities)
['Atlanta']
```

```
>>> cities = ['New York City', 3000000, 'Atlanta', 2000000, 'Detroit', 30000, '
New Orleans', 50000]
>>> foodDesertLocator(cities)
['New York City', 'Atlanta']
```

Grading Rubric

Function	Points
messageEncoder()	15
courseCreator()	20
compoundWords()	20
noteTaker()	25
foodDesertLocator()	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 resubmit 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.