Extra Credit HW

CS 1301 - Intro to Computing - Spring 2020

Important

- Due Date: **Monday, April 20th, 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.**

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
```

Important Note: The goal of this homework is to recall what you've learned throughout the year in order to solve new problems similar to those in past homeworks. Because this homework is an optional homework for extra credit, you will not be allowed to receive debugging assistance or specific function help from the TAs. However, you are still allowed to ask us conceptual or general questions.

Written by Arushi Gupta (arushigupta@gatech.edu) & Brae Davies (bdavies34@gatech.edu).

Lists & Tuples: TikTok Famous

Function Name: tiktok()
Parameters: list of sounds (`list` of `tuples`)
Returns: list of famous videos (`list` of `tuples`)
Description: To keep busy in quarantine, you decide you want to try to become TikTok famous. To have the best chance of getting famous, you want to make your videos based on popular videos that got a lot of likes. Given a `list` containing innerlists, each representing a sound and containing `tuples` with the creator name and number of likes, return a `list` of the most popular video for each sound ordered from most likes to least. If there are no videos for that sound, do not include it.

```
>>> print (tiktok([[("Charli", 45), ("Addison", 80), ("David", 12)],
                   [("Kyle", 3), ("Wyatt", 17)],
                   [("Sayler", 3), ("Cami", 12), ("Jules", 18), ("Brae", 1)]]))
[('Addison', 80), ('Jules', 18), ('Wyatt', 17)]
```

```
>>> print (tiktok([[("Peter", 17),("Arvin", 78),("Jakob", 2)],
                   [("Jasmine", 5)]]))
[('Arvin', 78), ('Jasmine', 5)]
```

Iteration: Note Passer

Function Name: notepassing()
Parameters: message (`str`), key word (`str`)
Returns: contains key word (`boolean`)
Description: You and your sibling decide that in order to pass time during quarantine, you want to start passing notes. To get across a certain message, you decide to write the key word backwards. Given a note (`str`) from your sibling and a key word (`str`) you are looking for, return the `boolean` stating whether the keyword is in the message.
Note: You may not use the `in` keyword.

```
>>> print (notepassing("imissthehcaeb", "beach"))
True
```

```
>>> print (notepassing("watcheivoomtime", "movie"))
False
```

Dictionaries: Scavenger Hunt

Function Name: scavengerhunt()
Parameters: list of items and their point values (`list` of `tuples`), list of people and what they found (`list` of `tuples`)
Returns: dictionary (`dict`)
Description: Your family decides to have a family scavenger hunt in the house. Write a function that takes in a `list` of `tuples` with items and their respective point values as well as a `list` of `tuples` containing people and a list of the items they found and returns a `dictionary` with the each name mapped to the person's total points.

```
>>> print (scavengerhunt([("wii remote", 7), ("baby picture", 10),
          ("trophy", 4)], [("Arvin", ["trophy"]), ("Arushi",["trophy",
          "wii remote"]), ("Brae", ["baby picture"])]))
{'Arushi': 11, 'Brae': 10, 'Arvin': 4}
```

```
>>> print (scavengerhunt([("card", 5), ("dog", 3), ("dollar bill", 9)],
           [("Tammie", ["dog", "dollar bill"]), ("Chad", ["card"])]))
{'Chad': 5, 'Tammie': 12}
```

Recursion: Constellations

Function Name: drawConstellations()
Parameters: rows (`int`), spaces (`int`)
Returns: None (`NoneType`)
Description: After a long day in the life of being quarantined, you go outside and lay down, looking at all the stars. While admiring the constellations, you decide you want to draw some of your own. Write a function that draws a pattern given a number of rows (`int`) the pattern must have. The first line in the pattern begins with `spaces` number of spaces. Each row adds an additional space. Your function should draw a triangle of asteriks by printing to the Shell. This must be done recursively. You may not use loops.
Note: If the pattern has zero rows, the function should print an empty string.

```
>>> drawConstellations(5,0)
*********
 *******
  *****
   ***
    *
```

```
>>> drawConstellations(3,5)
     *****
      ***
       *
```

```
>>> drawConstellations(2,10)
          ***
           *
```

OOP: Sandwich

Class Name: Sandwich
Description: After drawing out some constellations, you decide to close the night out with some dinner. You're craving... sandwiches! Write a class that represents a Sandwich object.
Note: The given `HWEC.py` has a repr method that we have already defined for you. Do not delete or change this as this is needed for testing.

Attributes:

- sauces (`list of str`): a list of sauces in the sandwich
- veggies (`list of str`) : a list of veggies in the sandwich
- breadSlices (`int`): number of slices of bread
- cheese(`int`): number of slices of cheese

Methods:

- __init__
  - initializes the following attributes:
    - sauces (`list of str`)
    - veggies (`list of str`)
    - cheese(`int`)
    - breadSlices (`int`) -- Default to 2 if no parameter is passed in
- __eq__
  - This method checks to see if two sandwiches are equal.
  - Two sandwiches are equal if their veggies contain the same elements (perhaps not in the same order) and they have the same number of bread slices.
- addVeggie
  - This method takes in a veggie (`str`) that you want to be added in your sandwich. If the veggie already exists in your sandwich, add another slice of cheese to your sandwich.
- merge
  - This method should take in two Sandwich objects (referred to as sandwich1 and sandwich2).

- This method will return a Sandwich object that combines sandwich 1's and sandwich 2's veggies and sauces. The new Sandwich's veggies and sauces should have no duplicates.
- The new sandwich should have half of the sum of sandwich 1's and sandwich 2's bread slices. The result should be an integer. (Hint: Use integer division)
- The new sandwich should have 2 slices of cheese.

Grading Rubric

Function	Points
tiktok()	15
notepasser()	20
scavengerhunt()	20
drawConstellations()	20
Sandwich - __init__	4
Sandwich - __eq__	5
Sandwich - addVeggie	8
Sandwich - merge	8
Total	**100**

Provided

The `HWEC.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 `HWEC.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 `HWEC.py` on Canvas.