Homework 9 - Recursion

CS 1301 - Intro to Computing - Fall 2020

Important

- Due Date: **Tuesday, November 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 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 recursion. The homework will consist of 5 functions for you to implement. You have been given `HW09.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:

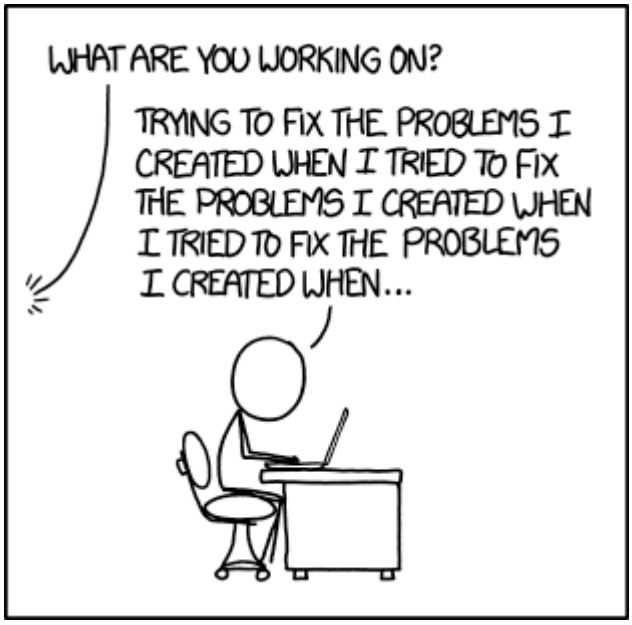
```
Test failed: False is not true
```

Written by Alexa O'Reilly (aoreilly@gatech.edu), Yara Seif (yaraseif@gatech.edu), and Michael Sullivan (michaelsullivan@gatech.edu)

Helpful Information to Know

All of these functions **must be implemented recursively**. You are not allowed to use loops in any of the functions.

Remember that recursion involves a **base case** and a **recursive call** inside of the function. You may need to include more than one base case and/or recursive call depending on the problem. Think about the solution to the smallest case (the base case) and then figure out how you can work your way down to the base case using recursive steps/calls!



xkcd.com

Picky Eater

Function Name: pickyEater()
Parameters: food list (`list`)
Returns: number of food items that can be eaten (`int`)
Description: You and your friends are throwing a Thanksgiving party and have invited a friend who is a picky eater. Your friend only eats food if it has an even number of characters. Given a list of the food that you will be serving, return the number of items that your friend can eat. If one of the items in the list is an empty string, do not count it as an even food.

```
>>> pickyEater(["Apple Pie", "Turkey", "Cranberry Sauce", "Mac and cheese"])
2
```

```
>>> pickyEater(["Pumpkin Pie", "Casserole", "Mashed Potato", "Pecan Pie"])
0
```

Invite Friends

Function Name: inviteFriends()
Parameters: list of invitees (`list`)
Returns: flattened list of all invitees (`list`)
Description: For your friendsgiving dinner, you want to make sure you are inviting everyone you know! You ask your friends who you should invite. Some friends give you a name of one invitee, some give you a list of invitees, and some even give you a list of lists containing invitees and their friends! To make things easy, write a function that takes in the nested list and returns a flattened list containing all of the names that your friends suggested.

```
>>> friendsList = [
        'Parul',
        'Megan',
        ['Arvin', 'Anthony', 'Arushi', ['Jasmine', 'Josh', 'Yara']]
    ]
>>> inviteFriends(friendsList)
['Parul', 'Megan', 'Arvin', 'Anthony', 'Arushi', 'Jasmine', 'Josh', 'Yara']
```

```
>>> friendsList = [
        'Alexa',
        [],
        ['Craig', 'Michael', 'Rajit'],
        ['Jakob', 'Damian']
    ]
>>> inviteFriends(friendsList)
['Alexa', 'Craig', 'Michael', 'Rajit', 'Jakob', 'Damian']
```

Friendsgiving Dinner

Function Name: friendsgiving()
Parameters: stores (`list`), budget (`float`), maxDistance (`int`)
Returns: price of sauce at each store (`dict`)
Description: As you are finishing up the preparations for the dinner, you realize you are missing the cranberry sauce. The rest of your friends will be showing up in 30 minutes, so you need to find the closest stores that have cranberry sauce. As a broke college kid, you are also on a budget and can't spend too much money. Write a function that takes in a list of tuples (`list`), where each tuple contains a **unique** store name, its price for cranberry sauce, and its distance from you. The function also takes in a budget (`float`), and a max allowable distance from your apartment (`int`), Return a dictionary of the possible stores that sell sauce for **less** than your budget and are **closer** than the max distance. The dictionary should have the store names as keys, and the price of their cranberry sauce as the values.

```
>>> stores = [('Whole Foods', 2.79, 4), ('Kroger', 6.99, 5)]
>>> budget = 4.99
>>> maxDistance = 6
>>> friendsgiving(stores, budget, maxDistance)
{'Whole Foods': 2.79}
```

```
>>> stores = [
        ('Sprouts', 3.45, 2),
        ('ALDI', 3.69, 6),
        ('Walgreens', 1.99, 1),
        ('Kroger', 2.79, 4)
    ]
>>> budget = 3.50
>>> maxDistance = 5
>>> friendsgiving(stores, budget, maxDistance)
{'Kroger': 2.79, 'Walgreens': 1.99, 'Sprouts': 3.45}
```

Palindrome Finder

Function Name: palindrome()
Parameters: string (`str`), guess (`int`)
Returns: Whether the string contains a number of palindromes (`bool`)
Description: You saw a meme about how **tacocat** is a palindrome and want to learn more about palindromes. However, when you try to guess the number of palindromes, you're not always correct. Write a function that takes in a string and a guess for how many 3 letter palindromes are in the string. The function should return whether this guess is correct. You can assume that the guess will never be negative.

Note: A 3 letter palindrome is a character surrounded by a different character on each side in the form **aba**. The pattern **aaa** does not count.

```
>>> palinstring = "tacocat" #palindrome is coc
>>> palindrome(palinstring, 1)
True
```

```
>>> palinstring = "cdecec" #palindromes are ece and cec
>>> palindrome(palinstring, 3)
False
```

Grading Rubric

Function	Points
pickyEater()	20
inviteFriends()	20
friendsgiving()	30
palindrome()	30
Total	**100**

Provided

The `HW09.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 `HW09.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 "Re-submit" button at the lower right-hand corner of Gradescope. You do not need to submit your `HW09.py` on Canvas.