Practice Problems

CS 1301 - Intro to Computing - Fall 2024

Important

- Resources:
  - TA Helpdesk
  - Email TA's or use Ed Discussion
  - Textbook: How to Think Like a Computer Scientist
  - CS 1301 YouTube Channel
  - Handouts (in Canvas Files)
- 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.**

Purpose

The purpose of the practice problems is to strengthen your coding skills and prepare you for the coding section of your exams. The problems are created to reflect the difficulty level of problems you might see on an exam. They will challenge you to combine and apply the different concepts you have learned in class. We will not be providing a skeleton file to assist you with completing the practice problems. Please name your file `practiceProblems.py` before submitting to Gradescope!

Grading

You are given the opportunity to earn 0.5% of extra credit applied to your final grade by successfully completing all of the practice problems. Practice problems will be progressively released throughout the semester, but in the end, there will be a total of 20 practice problems for 100 autograded points. The amount of extra credit (as a percent) you will earn towards your final grade can be calculated by:

```
extraCreditPercentage = autogradedPoints / 200
```

The deadline to submit practice problems for extra credit is **Tuesday, December 3rd at 11:59pm**.

Week 1: Functions and Expressions

Rock, Ramble, and Roll at the Student Center

Function Name: rockRambleRoll()
Parameters: N/A
Returns: None
Description: You have arrived at the Student Center for this year's Rock, Ramble, and Roll, and you are tasked with ordering food for your friends. At the Student Center, pizza at Campus Crust costs $9.00, sandwiches at 5th Street Deli cost $5.50 each, sushi rolls at Bento Sushi costs $8.00 each, and tacos at Twisted Taco cost $2.25 each. Write a function called `rockRambleRoll()` that asks the user how many pizzas, sandwiches, sushi rolls, and tacos you need to order for you and your friends. The function should then print the total cost of all of the food ordered.

Note: You can assume all inputs will be integers.

```
>>> rockRambleRoll()
How many pizzas do you want at Campus Crust? 5
How many sandwiches do you want at 5th Street Deli? 2
How many sushi rolls do you want at Bento Sushi? 1
How many tacos do you want at Twisted Taco? 7
You need to spend $79.75 at Rock, Ramble, and Roll.
```

```
>>> rockRambleRoll()
How many pizzas do you want at Campus Crust? 1
How many sandwiches do you want at 5th Street Deli? 0
How many sushi rolls do you want at Bento Sushi? 1
How many tacos do you want at Twisted Taco? 0
You need to spend $17.0 at Rock, Ramble, and Roll.
```

House Party at Bobby Dodd Stadium

Function Name: houseParty()
Parameters: N/A
Returns: None
Description: You are excited to visit Bobby Dodd Stadium for their annual house party, but you need to make sure to leave enough time to visit the evening mixer at the exhibition hall. Each activity at the house party takes a set amount of time: each autograph takes 10 minutes, each visit to the bouncy castle takes 20 minutes, each video game contest takes 45 minutes, and walking through Georgia Tech's locker room takes 15 minutes. Write a function called `houseParty()` which asks the user how many times the activities listed above will be done. This function should then print out the total time in hours and minutes the user will spend at the house party.

Note: You can assume all inputs will be integers.

```
>>> houseParty()
How many autographs will you get? 5
How many times will you visit the bouncy castle? 1
How many video game contests will you participate in? 0
How many times will you walk through the locker rooms? 1
You will spend 1 hour(s) and 25 minutes at the house party.
```

```
>>> houseParty()
How many autographs will you get? 6
How many times will you visit the bouncy castle? 3
How many video game contests will you participate in? 0
How many times will you walk through the locker rooms? 4
You will spend 3 hour(s) and 0 minutes at the house party.
```

Week 2: Conditionals

Open Sesame!

Function Name: enterTheCave()
Parameters: phrase (`str`), code (`int`)
Returns: None (`NoneType`)
Description: You are part of a group of hackers who have a secret hideout in a cave. A few months ago, your passcode was leaked, and a civilian managed to break in. You have been asked to improve the security of the cave by adding another layer of complexity to the cave's entrance system. Write a function that takes in a phrase `(str)` and a code `(int)`. If the phrase is `"Open Sesame"` and the code is between 0 and 25 (both inclusive), then you open the door. This means the function should **print** `"You have opened the door"`. The door can also be opened with the phrase `"Hello World"` and the code between 75 and 100 (both inclusive). If the phrase is `"Python Enjoyer"` and the code is exactly 50, the function should **print** `"You have closed the door"`. If the phrase and code combination is outside of these possibilities, the function must **print** `"INTRUDER ALERT"`.

```
>>> enterTheCave("Open Sesame", 22)
You have opened the door
```

```
>>> enterTheCave("Hello World", 50)
INTRUDER ALERT
```

Dinner Time

Function Name: dinnerTime()
Parameters: xCoordinate (`int`)
Returns: closestRestaurant (`str`)
Description: The sun is setting and it is time for you and your friends to grab dinner at the Bright Beach Boardwalk. Since you have already been out and about the entire day, your group decides to eat at the closest nearby restaurant. Write a function that takes in your group's current x-coordinate on Bright Beach Boardwalk. This function should return the restaurant that is the closest on the boardwalk to your group. Reference the table below for the locations of various restaurants on Bright Beach Boardwalk.

Note: You can assume there will be no ties.

Restaurant	X-Coordinate
Sun, Sand, and Seafood	250
The Beachcomber's Bistro	460
Coastal Catch	820
Beachside Bonanza	870
Tidal Wave Tavern	940

```
>>> dinnerTime(800)
'Coastal Catch'
```

```
>>> dinnerTime(310)
'Sun, Sand, and Seafood'
```

Week 3: Iteration

Upside Down Light Communicator

Function Name: lightCommunicator()
Parameters: upsideDownMessage (`str`)
Returns: message (`str`)
Description: To communicate with your friends in the Upside Down, you need to reverse the messages you are receiving as well as flip the S's and the T's (case insensitively). Given the message you have received, return the translated message.

```
>>> lightCommunicator("!EDIH - gnimoc ti nogrogomed ehS")
'The demogorgon is coming - HIDE!'
```

```
>>> lightCommunicator("!emoh emoc ,lliW")
'Will, come home!'
```

Binge Time

Function Name: totalTime()
Parameters: episodeRuntimes (`str`)
Returns: totalWatchTime(`str`)
Description: The new season of Ginny and Georgia just came out and you cannot wait to catch up on all the drama! Given a string of episode run times (each separated by a space), return the number of hours and minutes it will take you to binge all episodes as a string. The returned string should be of the form `"{} hour(s) and {} minute(s)"`, and you may assume you will always have a whole number for the number of minutes.

```
>>> totalTime("87 100 70")
'4 hour(s) and 17 minute(s)'
```

```
>>> totalTime("50 66 90 43")
'4 hour(s) and 9 minute(s)'
```

Week 4: String Manipulation

Meatless Monday

Function Name: meatlessMonday()
Parameters: dishes (`str`)
Returns: vegetarianFoods (`list`)
Description: It's Meatless Monday in Willage, and you've got the job of labeling all the food. Unfortunately, all the name cards are mixed up. Write a function that takes a string containing all the names of all the dishes, separated by dashes. All non-alphabetic characters need to be removed from the names. If the name of the dish begins with the letter `'V'` (case insensitive), it is vegetarian. Return a list of all the vegetarian foods, with the `'V'` removed, **sorted in alphabetical order**. If there are no foods in the dining hall, this function should return an empty list.

Note: The returned list should have all vegetarian foods in lowercase.

```
>>> meatlessMonday('l@am$b-vr@ice-Vbe!ans-po##rk')
['beans', 'rice']
```

```
>>> meatlessMonday('t!a0cos-vpi!zz#a-ChiC!ke#@n-vcAu!li$flo!wer')
['cauliflower', 'pizza']
```

Week 5: Lists

Eating Out

Function Name: eatingOut()
Parameters: yourList (`list`), friendList (`list`)
Returns: sharedRestaurants (`list`)
Description: It hasn't even been half way through the semester yet, and you and your friend have already gotten sick of Georgia Tech's dining halls. You have decided to eat out for a change, but it is hard to decide where to go. Given lists of you and your friend's favorite restaurants, write a function that returns a list of only the restaurants that are in **both** of your favorite restaurant lists, **sorted in alphabetical order**. If there are no restaurants that are on both of your lists, return the string `"Whatever, we'll go to Nave."` .

```
>>> yourList = ['Gyro Bros', 'Subway', 'Ponko Chicken', 'Tin Drum']
>>> friendList = ['McDonalds', "Umma's", 'Subway', 'Gyro Bros']
>>> eatingOut(yourList, friendList)
['Gyro Bros', 'Subway']
```

```
>>> yourList = ['Blue Donkey', "Kaldi's", 'Panda Express', 'Halal Guys']
>>> friendList = ['5th Street Deli', 'Brain Freeze', 'Panda Express']
>>> eatingOut(yourList, friendList)
['Panda Express']
```

Week 6: Tuples and Modules

Rocky Racers

Function Name: organizeTimes()
Parameters: unorganizedTimeList (`list`)
Returns: organizedTimeList (`list`)
Description: While trying to find the bathroom in the CRC, you run into the rock climbing club. Before you can get away, they ask you to be the referee in their rock climbing contest. You need to organize their data containing the times it takes each climber to scale their wall.

Given a list `namesAndTimes` that contains tuples of the form `([time1, time2, time3], name)`, write a function `organizeTimes()` that reorganizes the list and removes the slowest time of each climber. Return the reorganized list. Make sure that the returned list is sorted **alphabetically** by name, and that the inner lists of times are organized by **fastest to slowest**.

```
>>> unorganizedTimeList = [([7.05, 8.0, 6.2], "Andre"),
                            ([6.14, 9.2, 7.2], "Collin"),
                            ([7.42, 30.2, 6.34], "Ethan"),
                            ([9.2, 8.1, 6.23], "Evan")]
>>> organizeTimes(unorganizedTimeList)
[('Andre', [6.2, 7.05]), ('Collin', [6.14, 7.2]), ('Ethan', [6.34, 7.42]),
 ('Evan', [6.23, 8.1])]
```

```
>>> unorganizedTimeList = [([27.52, 28.54, 16.22], "Naomi"),
                            ([36.49, 48.32, 21.47], "Chris")]
>>> organizeTimes(unorganizedTimeList)
[('Chris', [21.47, 36.49]), ('Naomi', [16.22, 27.52])]
```

Do You Even Bench?

Function Name: findPlates()
Parameters: maxBench (`int`)
Returns: numPerPlate (`list`)
Description: It's time to bench (barbell bench press) at the CRC. You are going for a new personal record, but you do not know which plates to put on the bar! Fortunately your super buff and strong gym bros have created a handy-dandy module that will help calculate this for you.

Write a function called `findPlates` that takes in a weight `maxBench` and returns a list of tuples in the format `(plateWeight, numOfPlates)` that equates to the weight you need to add to the barbell, in the order of decreasing weight. Use the function `plateCalculator` in the gymBroTools.py module to help you (see documentation below for more details). There are 5 types of plates, each weighing 45, 35, 25, 10, and 5 pounds, but there are only 5 of each type of plate. If the `maxBench` is over 645, less than or equal to 45 pounds, or not divisible by 5, return an empty list.

Note: `plateCalculator()` already takes into account the weight of the bar (45 pounds). Therefore `plateCalculator()` will provide you with the quantity and type of plates required **in addition** to the bar's weight. The list you return should also not include the weight of the bar.
Note: Remember to place gymBroTools.py in the same folder as practiceProblems.py.

```
>>> findPlates(440)
[(45, 5), (35, 4), (25, 1), (10, 0), (5, 1)]
```

```
>>> findPlates(325)
[(45, 5), (35, 1), (25, 0), (10, 2), (5, 0)]
```

gymBroTools — Documentation

gymBroTools.plateCalculator(weight)

From an integer `weight` passed in as an argument, `plateCalculator()` returns a tuple of strings corresponding to the plates needed. It will also subtract the weight of the bar (45 pounds) to tell you how exactly many **plates** you need to add to the bar. If the weight is greater than 645 pounds or not divisible by 5, an empty tuple will be returned.

```
>>> import gymBroTools
>>> gymBroTools.plateCalculator(440)
('45', '45', '45', '45', '45', '35', '35', '35', '35', '25', '5')
```

Week 7: Dictionaries

Lunch Spots

Function Name: lunchSpots()
Parameters: friendsPreferences (`dict`)
Returns: spotPreferences (`dict`)
Description: You and your friends want to find a place on campus to have lunch together. Given friendsPreferences, which is a dictionary that maps a friend's name to the list of locations they prefer, write a function that will return a dictionary that maps a location to a list of all the friends who prefer it. Make sure the elements of the lists in the dictionary are sorted **alphabetically**.

```
>>> friendsPreferences = {
    "Andrew": ["Klaus", "CoC", "Bunger-Henry"],
    "Chris": ["CODA", "Scheller", "Klaus", "CoC", "Howey"],
    "Cynthia": ["CULC", "Klaus", "Boggs"],
    "Audrey": ["Klaus", "Scheller", "Boggs", "Howey"],
    "Priya": ["Klaus", "CoC", "CULC"],
    "Josh": ["CULC", "CoC", "Bunger-Henry"]
}
>>> lunchSpots(friendsPreferences)
{'Boggs': ['Audrey', 'Cynthia'],
 'Bunger-Henry': ['Andrew', 'Josh'],
 'CODA': ['Chris'],
 'CULC': ['Cynthia', 'Josh', 'Priya'],
 'CoC': ['Andrew', 'Chris', 'Josh', 'Priya'],
 'Howey': ['Audrey', 'Chris'],
 'Klaus': ['Andrew', 'Audrey', 'Chris', 'Cynthia', 'Priya'],
 'Scheller': ['Audrey', 'Chris'],
}
```

```
>>> friendsPreferences = {
    "Paige": ["Klaus", "Scheller", "Howey"],
    "Ramya": ["Scheller", "Howey"],
    "Harshith": []
}
>>> lunchSpots(friendsPreferences)
{'Howey': ['Paige', 'Ramya'],
 'Klaus': ['Paige'],
 'Scheller': ['Paige', 'Ramya']
}
```

Credit Hour Total

Function Name: creditHours()
Parameters: courseCatalog (`dict`), myCourses (`dict`)
Returns: creditHoursTotal (`int`)
Description: It's never too early to plan out your course load for next semester! Given courseCatalog (`dict`) and myCourses (`dict`), write a function that returns the total number of credit hours you are taking next semester. The course catalog dictionary maps the department (`str`) to a dictionary containing course numbers (`int`) that are mapped to their respective credit hours (`int`). The courses you plan to take is also a dictionary that contain departments (`str`) mapped to a list of course numbers (`list`). You may assume that courses you plan to take will always be found in the course catalog.

```
>>> courseCatalog = {
  'CS': {1301: 3, 2200: 4, 2340: 3, 3001: 3},
  'MATH': {2550: 2, 2552: 4, 3012: 3, 3670: 3},
  'EAS': {1600: 4, 1601: 4, 2600: 4}
}
>>> myCourses = {
  'CS': [2200, 2340, 3001],
  'MATH': [3012],
  'EAS': [2600]
}
>>> creditHours(courseCatalog, myCourses)
17
```

```
>>> courseCatalog = {
  'CS': {1301: 3, 2110: 4, 2340: 3, 3510: 3, 3600: 3},
  'MATH': {2550: 2, 2552: 4, 3012: 3, 3670: 3},
  'PHYS': {2211: 4, 2212: 4, 2231: 5, 2232: 5}
}
>>> myCourses = {
  'CS': [2110, 2340, 3510, 3600],
  'MATH': [3670],
  'PHYS': [2232]
}
>>> creditHours(courseCatalog, myCourses)
21
```

Week 8: File I/O

movies.txt

The `movies.txt` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and move the file to the same folder as your `practiceProblems.py` file. To open the `.txt` file, you can use the built-in Notepad for Windows, TextEdit for Mac, or any other text editor of your choice.

You will be working with a text file that contains data about movies. The data will contain a movie title, country of origin, release year, and whether it's been seen before by your father (Watched / Not Watched). Each movie's data will be separated by a newline. The code below shows how the text file will be formatted. See the provided `movies.txt` file as an example.

Note: `movies.txt` will never contain duplicate movies.

```
Movies:

Movie #1 Title
Movie #1 Country
Movie #1 Release Year
Movie #1 Seen Before

Movie #2 Title
Movie #2 Country
Movie #2 Release Year
Movie #2 Seen Before

Movie #3 Title
Movie #3 Country
Movie #3 Release Year
Movie #3 Seen Before

...
```

Foreign Films

Function Name: foreignFilms()
Parameters: friendTupList (`list`)
Returns: friendDict (`dict`)
Description: Your international friends are coming over and you want to watch some foreign films together. Write a function that takes in a list of tuples, where the first element in the tuple is a friend's name, and the second element is his/her home country. Your function should return a dictionary that maps your friends' names to a list of movies from their country. Use the `movies.txt` file to find what country each movie is from. If you can't find any movies for a certain friend, their list should be empty.

Note: The movies within each friend's list should be sorted alphabetically.

```
>>> foreignFilms([('Taj', 'India'), ('Sam', 'South Korea'), ('Kailen', 'Mexico')])
{'Taj': ['Barfi', 'Lagaan'],
 'Sam': ['Parasite', 'Train to Busan'],
 'Kailen': ['Como Agua Para Chocolate', 'Roma']}
```

```
>>> foreignFilms([('Chisom', 'Nigeria'), ('Malhar', 'Singapore'), ('Nita', 'Italy')]
{'Chisom': [], 'Malhar': [], 'Nita': ['Life is Beautiful']}
```

Week 9: CSVs

For this part of the practice problems, the `.csv` file being read from will be formatted as shown below. Be sure to download the provided file from Canvas, and move it into the same folder as your `practiceProblems.py` file.

By default, your computer will likely use Excel on Windows or Sheets on Mac to open the `.csv` file, but don't be alarmed: reading values from a `.csv` file is no different than a `.txt` file; the only difference lies in the formatting of the data.

The summerDrinks CSV file (and other `.csv` files like it) will contain a beverage name, the best temperature in which to drink it, whether it's caffeinated or not, and the month in which the ingredients are in season. Each data entry will be separated by a newline. Below is an example of how the `.csv` file will be formatted. Note that the first row contains the headers for each of the columns, and does not contain any actual data.

Format of `summerDrinks.csv`:

```
name,temperature,caffeinated,month
name1,temperature1,caffeinated1,month1
name2,temperature2,caffeinated2,month2
...
```

Seasonal Drinks

Function Name: inSeason()
Parameters: favDrinks (`list`), currentMonth (`str`)
Returns: inSeasonDrinks (`list`)
Description: Many drinks can be cheaper if their ingredients are in season, and as a broke college student, you want to save as much money as you can. So, given a list of your favorite drinks and the current month, write a function that returns a list of the drinks that are currently in season, **sorted in alphabetical order**. Additionally, since you must wake up early to your 8:25 AM classes, make sure none of the drinks in the list you return are caffeinated.

Note: If there are no drinks that are in season then return `"Nothing is in season :("`.

```
>>> favDrinks = ["cappuccino", "lychee passionfruit smoothie",
                "oolong tea", "london fog"]
>>> inSeason(favDrinks, "September")
"Nothing is in season :("
>>> favDrinks = ["matcha", "pina colado smoothie", "coconut smoothie", "acai"]
>>> inSeason(favDrinks, "July")
["acai", "coconut smoothie"]
```

Week 10: APIs

For this part of the practice problems, we will be using the REST countries API (https://restcountries.com). Please read through the documentation, as it will be extremely helpful for completing this part.

For all of your requests, make sure to use version 3.1 of the REST countries API (V3.1), not version 2 (V2).

If you make a valid request with the URL: https://restcountries.com/v3.1/alpha/usa, you will receive the following JSON formatted response:

```
[{
    "name": ...,
    "tld": [".us"],
    "cca2":"US",
    "ccn3":"840",
    "cca3":"USA",
    ...
}]
```

If you make a request with an invalid URL, you will receive the following response:

```
{
  "status": 400,
  "message": "Bad Request"
}
```

Note: Do NOT use the `mock_requests` module for this section of the practice problems. Please use the standard `requests` module.

Neighbors

Function Name: shareBorder()
Parameters: country1 (`str`), country2 (`str`)
Returns: areNeighbors (`bool`)
Description: How can we check if two countries are neighbors? Write a function that uses the restcountries API to check whether two countries passed in as parameters are neighbors or not. Your function should return `True` if the two countries share a border, and `False` if they don't.

Note: A country does not share a border with itself.
Hint: If a country has no borders, the data pertaining to the country will not have a "borders" key.

```
>>> shareBorder("United States of America", "Canada")
True
```

```
>>> shareBorder("People's Republic of China", "Japan")
False
```

Currency Popularity

Function Name: currencyRatio()
Parameters: continent (`str`), currency (`str`)
Returns: ratio (`float`)
Description: You're planning to go on vacation, but you only have one type of currency. Given the name of a continent and a currency, return the ratio of the number of countries in which your chosen currency is official to the total number of countries in the continent **rounded to 2 decimal places**. You can assume that both the continent and currency will be valid.

Note: The currency passed in will be the currency's common name, **not** its currency code.

```
>>> currencyRatio("Europe","Euro")
0.51
```

```
>>> currencyRatio("Americas","United States dollar")
0.2
```

Week 12: Recursion

All problems in this section must be completed with recursion. No credit will be awarded for solutions that use loops.

Ramya's Running Out!

Function Name: numCoffees()
Parameters: num (`int`)
Returns: None (`NoneType`)
Description: Ramya is a barista working the closing shift at Blue Donkey, and she only has a limited number of coffees left to sell. Given the starting number of coffees (`int`), write a function that recursively prints out the number of coffees left in the format `"Coffees left: {num}"`. Once there are no coffees left, print the string `"No more coffee!"`.

Hint: If the passed in value is not positive, immediately print `"No more coffee!"`.

```
>>> numCoffees(4)
Coffees left: 4
Coffees left: 3
Coffees left: 2
Coffees left: 1
No more coffee!
```

```
>>> numCoffees(1)
Coffees left: 1
No more coffee!
```

Cynthia's Recommendations

Function Name: atlCoffee()
Parameters: listOfshops (`list`)
Returns: shopsToVisit (`dict`)
Description: Hell week is right at your doorstep, and you can't stand the idea of studying on campus anymore. Luckily, Cynthia has got your back! She gives you a list containing the names of coffee shops she's been to in Atlanta, the area they're located in, and a rating out of 10. Since you're short on time, you decide to visit only the places that have a ranking greater than 5. Write a recursive function that takes in a list of tuples containing the name, area, and rating of the shops Cynthia likes and returns a dictionary of the places with a rating greater than 5 arranged by area and sorted in alphabetical order. If all the places are ranked too low, return an empty dictionary.

Hint: Sort the elements inside the lists in alphabetical order, not the dictionary itself. Recall that dictionaries are unordered.

```
>>> atlCoffee([('Chattahoochee CoffeeCo','West Midtown',8),
               ('Brash','West Midtown',7),
               ('Spiller Park','Toco Hills',7)])
{'Toco Hills': ['Spiller Park'],
 'West Midtown': ['Brash', 'Chattahoochee CoffeeCo']}
```

```
>>> atlCoffee([('Chrome Yellow','Sweet Auburn',7),
               ('Little Tart','Oakland',10),
               ('Cold Brew Bar','Oakland',10)]
{'Oakland': ['Cold Brew Bar', 'Little Tart'],
 'Sweet Auburn': ['Chrome Yellow']}
```

Week 13: OOP

You and your friends are planning a potluck! In this part of the practice problems, you will be implementing two classes: `Friend` and `Potluck` based on the following descriptions.

Friend

Attributes:

- name (`str`): name of friend
- age (`int`): age of friend
- foodItem (`str`): item that friend is bringing to the potluck
- rsvpStatus (`bool`): value indicating whether the friend will come or not

Methods:

- __ init __
  - Initializes the following attributes
    - name (`str`)
    - age (`int`)
    - foodItem (`str`) - if no foodItem value is passed in, set the attribute to a default value of `None`
    - rsvpStatus (`bool`) - if no RSVP status is passed in, set the attribute to a default value of `True`
- __ str __
  - This method should create a string representation of the `Friend` object in the format of: `"My name is {name}. I am {age} years old, and I am bringing {foodItem}!"` if that person's RSVP status is True. Otherwise, this method should create a string representation in the format of: `"My name is {name}. Unfortunately, I cannot go to the potluck :("`
- __ eq __
  - Two `Friend` objects are equal if and only if they have the same `name` , `age` , and `foodItem` attributes.

Potluck

Attributes:

- host (`Friend`): a Friend object hosting the potluck
- guestList (`list`): list containing all Friend objects invited to the potluck

Methods:

- __ init __
  - Initializes the following attributes:
    - host (`Friend`)
    - guestList (`list`) - Should **always** be initialized to an empty list.
- __ str __
  - This method should create a string representation of the `Potluck` object in the format of: `"My name is {hostName}, and I am hosting a potluck. There are {numGuests} people coming."` .
- invite()
  - This method should take in a guest (Friend object) as a parameter.
  - If the guest's RSVP status is True, add a two-item tuple to the guest list that contains their name and food item they are bringing to the potluck.
  - Do nothing if the guest's RSVP status is False.
- checkFood()
  - This method does not take in any additional parameters.
  - Go through the guest list and check if every guest is bringing a valid food item i.e. that item is not None. If everyone is bringing a valid food item, the method should print `"Nice! Everyone brought something!"` . However, if at least one guest is not bringing a valid food item, print `"Uh oh! {name} forgot to bring food!"` for each one of these guests.

OOP Test Cases

The OOP Test Cases are described here in detail to assist with your debugging with the autograder.

OOP Test 000

Name: Create Friend
Points: 0.5
Prerequisites: None
Objects Created:

```
avinash = Friend('Avinash', 20, "cookies")
```

Methods Called: None
Tests:

- Attributes of `avinash`: name = 'Avinash', age = 20, foodItem = 'cookies', rsvpStatus = True

OOP Test 001

Name: Create Friend
Points: 0.5
Prerequisites: None
Objects Created:

```
chris = Friend('Chris', 20, 'apple juice')
```

Methods Called: None
Tests:

- Attributes of `chris`: name = 'Chris', age = 20, foodItem = 'apple juice', rsvpStatus = True

OOP Test 002

Name: Create Friend
Points: 0.5
Prerequisites: None
Objects Created:

```
shayan = Friend('Shayan', 19, None, False)
```

Methods Called: None
Tests:

- Attributes of `shayan`: name = 'Shayan', age = 19, foodItem = None, rsvpStatus = False

OOP Test 003

Name: Create Friend
Points: 0.5
Prerequisites: None
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
```

Methods Called: None
Tests:

- Attributes of `teghpreet`: name = 'Teghpreet', age = 19, foodItem = None, rsvpStatus = True

OOP Test 004

Name: Friend String Method
Points: 0.5
Prerequisites: OOP Test 000
Objects Created:

```
avinash = Friend('Avinash', 20, "cookies")
```

Methods Called:

```
print(avinash)
```

Output:

```
My name is Avinash. I am 20 years old, and I am bringing cookies!
```

OOP Test 005

Name: Friend String Method
Points: 0.5
Prerequisites: OOP Test 001
Objects Created:

```
chris = Friend('Chris', 20, 'apple juice')
```

Methods Called:

```
print(chris)
```

Output:

```
My name is Chris. I am 20 years old, and I am bringing apple juice!
```

OOP Test 006

Name: Friend String Method
Points: 0.5
Prerequisites: OOP Test 002
Objects Created:

```
shayan = Friend('Shayan', 19, None, False)
```

Methods Called:

```
print(shayan)
```

Output:

```
My name is Shayan. Unfortunately, I cannot go to the potluck :(
```

OOP Test 007

Name: Friend String Method
Points: 0.5
Prerequisites: OOP Test 003
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
```

Methods Called:

```
print(teghpreet)
```

Output:

```
My name is Teghpreet. I am 19 years old, and I am bringing None!
```

OOP Test 008

Name: Friend Equals Method
Points: 0.5
Prerequisites: OOP Test 000
Objects Created:

```
avinash1 = Friend('Avinash', 20, "cookies")
avinash2 = Friend('Avinash', 20, "cookies")
```

Methods Called:

```
>>> avinash1 == avinash2
```

Output:

```
True
```

OOP Test 009

Name: Friend Equals Method
Points: 0.5
Prerequisites: OOP Test 000
Objects Created:

```
avinash1 = Friend('Avinash', 20, "cookies")
avinash2 = Friend('Avinash', 20, None)
```

Methods Called:

```
>>> avinash1 == avinash2
```

Output:

```
False
```

OOP Test 010

Name: Create Potluck
Points: 1
Prerequisites: OOP Test 003
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
potluck = Potluck(teghpreet)
```

Methods Called: None
Tests:

- Attributes of `potluck`: host = teghpreet, guestList = []

OOP Test 011

Name: Potluck String Method
Points: 1
Prerequisites: OOP Test 010
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
potluck = Potluck(teghpreet)
```

Methods Called:

```
print(potluck)
```

Output:

```
My name is Teghpreet, and I am hosting a potluck. There are 0 people coming.
```

OOP Test 012

Name: Potluck Invite Method
Points: 1
Prerequisites: OOP Test 000, OOP Test 001, OOP Test 002, OOP Test 003, OOP Test 010
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
avinash = Friend('Avinash', 20, "cookies")
chris = Friend('Chris', 20, 'apple juice')
shayan = Friend('Shayan', 19, None, False)
potluck = Potluck(teghpreet)
```

Methods Called:

```
potluck.invite(avinash)
potluck.invite(chris)
potluck.invite(shayan)
```

Tests:

- Attributes of potluck: host = teghpreet, guestList = [('Avinash', 'cookies'), ('Chris', 'apple juice')]

OOP Test 013

Name: Potluck Check Food Method
Points: 1
Prerequisites: OOP Test 000, OOP Test 001, OOP Test 002, OOP Test 003, OOP Test 010
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
avinash = Friend('Avinash', 20, "cookies")
chris = Friend('Chris', 20, 'apple juice')
shayan = Friend('Shayan', 19, None, False)
potluck = Potluck(teghpreet)
```

Methods Called:

```
potluck.invite(avinash)
potluck.invite(chris)
potluck.invite(shayan)
potluck.checkFood()
```

Tests:

- Attributes of potluck: host = teghpreet, guestList = [('Avinash', 'cookies'), ('Chris', 'apple juice')]

Output:

```
Nice! Everyone brought something!
```

OOP Test 014

Name: Potluck Check Food Method
Points: 1
Prerequisites: OOP Test 000, OOP Test 001, OOP Test 002, OOP Test 003, OOP Test 010
Objects Created:

```
teghpreet = Friend('Teghpreet', 19, None, True)
avinash = Friend('Avinash', 20, None)
chris = Friend('Chris', 19, None)
potluck = Potluck(teghpreet)
```

Methods Called:

```
potluck.invite(avinash)
potluck.invite(chris)
potluck.checkFood()
```

Tests:

- Attributes of potluck: host = teghpreet, guestList = [('Avinash', None), ('Chris', None)]

Output:

```
Uh oh! Avinash forgot to bring food!
Uh oh! Chris forgot to bring food!
```