CS 112 – Fall 2020 – Programming Assignment 8

Functions

Due Date: Monday, October 26th, 11:59pm

See the **"assignment basics"** file for more detailed information about getting assistance, running the test file, grading, commenting, and many other extremely important things. Each assignment is governed by the rules in that document.

Background

The purpose of this assignment is to practice writing function definitions, and using functions inside of other functions to solve complex problems.

Restrictions

Any function that violates any of the following will receive **zero** points even if it passes all tests.

- You are **not** allowed to `import` anything
- You are **not** allowed to use the `global` keyword, nor should you have any global variables.
    - In other words, **do not** create any variables outside of a function definition.
- You are **not** allowed to use slicing
- You are **not** allowed to use `sets` or `dictionaries`
- You are **not** allowed to use anything that hasn't been covered in class
- No built-in function except `range()` and `len()` is allowed
- From list methods, you are allowed to use `.append(), .insert(), .remove() or del`. Please **do not ask on piazza** whether you can use `.sort()`, `sorted()`, `.index()`, `.count()` etc.

Testing

From this assignment on, testing will be done a bit different than before. The grader we use to grade your work will be different from the tester we provide. This means that there will be cases that the tester doesn't check and, therefore, your code might have logic errors that the tester doesn't detect. You must do additional checks, on your own, to make sure that you haven't missed anything and your code is correct. You do ***not*** need to modify the tester. The goal is to help you put more focus on writing logically correct programs instead of trying to pass certain tests only. Thus, the grader (a.k.a. autograder) will be made available on Gradescope a few days after the release of this assignment, and will include '**hidden**' test cases.

Grading Rubric

Submitted correctly:	2	# see assignment basics file for file requirements!
Code is well commented:	8	# see assignment basics file for how to comment!
Autograder:	90	# see assignment basics file for how to test!
TOTAL:	100	

Note: If your code does not run and crashes due to errors, it will receive **zero** points. Turning in running code is essential. **Do not submit code with uncommented print statements**

Functions

In this assignment, you will be writing and calling a handful of functions to help you win the next time that you play Among Us. In this game, you find yourself on a broken ship with a crew. You must work together to complete tasks together to fix the ship. There is an imposter among your crew, wo will try to kill everyone before you can fix the ship! Figure out who the imposter is and eject them, or fix the ship, before they kill everyone!

The signature of each function is provided below, do ***not*** make any changes to them otherwise the tester will not work properly. The following are the functions you must implement:

`clearable(locations, report_location)` **[15pts]**

Description: There was a murder at **`report_location`**! Every player in the game has determined the location where they last saw a particular player. Determine if this player can be cleared of the murder based on this information. You can clear the player if **more than** 50% of the group saw them in a different location from the reported location, otherwise, they cannot be cleared!

Parameters: **`locations`** is a `list` of variable size, that contains strings. these strings are the name of different locations in the game. **`report_location`** is a single string, the name of a location where a murder was…. reported

Assumptions:

- Just because a location was reported doesn't mean that it will be in the list of last known locations.
- Remember, you **cannot** just use **`locations.count()`**

Return value: a Boolean; **`True`** if they can be cleared and **`False`** otherwise

Examples:

```
ex_loc = ['medbay','navigation','navigation','medbay','cafeteria','medbay']

# can you clear this player from the cafeteria?
clearable(ex_loc, 'cafeteria')  →   True
# yes, because 1 player saw them there, and 5 didn't! 1/6 is less than 5/6

# can you clear this player from navigation?
clearable(ex_loc, 'navigation') →   True
# yes, because 2 players saw them there, and 4 didn't! 2/6 is less than 4/6

# can you clear this player from medbay?
clearable(ex_loc, 'medbay')     →   False
# no! 3 players saw them there, and 3 didn't! 3/6 is not less than 3/6
```

`get_sightings(location_log, player)` [15pts]

Description: This returns a list of all locations in which player was sighted based on the location log, which contains information on where the last time each player saw every other player in a game of Among Us.

Parameters: **`location_log`** is a `list` of tuples, each containing 3 strings described below. **`player`** is the name of a player you're interested in

The entry in the **`location_log`** looks like **`(playerA, playerB, location)`** which means...

The last time **`playerA`** saw **`playerB`** was in **`location.`**

Assumptions:

- All inner tuples will be of length 3.
- **loc_log** will contain every pair of players and where they saw each other last
- playerA seeing playerB in one location does not imply that playerB has seen playerA there.
    - For example, **`('red', 'green', 'medbay')`** might exist in the log but **not** **`('green', 'red', 'medbay')`**

Return value: a list of all the location names where a particular player was seen. Order does not matter.

Examples:

```
ex_loc_log = [
              ('red', 'blue', 'medbay'),
              ('red', 'green', 'navigation'),
              ('green', 'blue', 'medbay'),
              ('blue', 'red', 'navigation'),
              ('green', 'red', 'cafeteria'),
              ('blue', 'green', 'electrical'),
          ]


# where was the 'red' player last seen?
get_sightings(ex_loc_log, 'red')     →    ['navigation', 'cafeteria']

# where was the 'green' played last seen?
get_sightings(ex_loc_log, 'green')   →    ['navigation', 'electrical']

# where was the 'blue' player last seen?
get_sightings(ex_loc_log, 'blue')    →    ['medbay', 'medbay']
```

`likely_imposters(report_location, players, location_log)` **[20pts]**

Description: Determine which players are likely to be the imposter given what is known about where people were last seen. Construct a list of all players who cannot be cleared of the murder at the reported location.

You must **correctly** call your **`clearable`** and **`get_sightings`** functions within the definition of this function to earn full credit!

Parameters: **`report_location`** is a single string, the name of a location where a murder was…. reported
`players` is a list of string play names, representing all players in the game
`location_log` is as described in **`get_sightings`**

Assumptions:

- All assumptions from **`get_sightings`** are valid here**.**
- **players** will be a complete list of players
- **loc_log** will contain every pair of players and where they saw each other last

Return value: A list of string player names who cannot be cleared of the murder at this given location. Order does not matter.

Examples:

```
ex_loc_log = [
              ('red', 'blue', 'medbay'),
              ('red', 'green', 'navigation'),
              ('green', 'blue', 'medbay'),
              ('blue', 'red', 'navigation'),
              ('green', 'red', 'cafeteria'),
              ('blue', 'green', 'electrical'),
              ('orange', 'blue', 'medbay'),
              ('orange', 'red', 'electrical'),
              ('blue', 'orange', 'medbay'),
              ('orange', 'green', 'navigation'),
              ('green', 'orange', 'navigation'),
              ('red','orange','medbay')
          ]

ex_players = ['red', 'green', 'blue','orange']

# 3/3 other players saw blue, and 2/3 other players saw orange, in medbay
likely_imposters('medbay', ex_players, ex_loc_log)          ->    ['blue', 'orange']

# no single player was seen enough times in the cafeteria
likely_imposters('cafeteria', ex_players, ex_loc_log))      ->    []    #empty list!

# 2/3 other players saw green in navigation
print(likely_imposters('navigation', ex_players, ex_loc_log))     ->    ['green']
```

`task_filter(task_log, status=None, task_type=None, *ignored_names)` **[20pts]**

Description: Construct a list of task names that match a given criteria based on its status and type. Do not include the names of any specifically named tasks after these criteria.

Parameters: The **`task_log`** will be a list of tuples with 3 elements each. Those elements are:

`(name, status, type)` where…
- **`name`** is a string that represents the name of this task,
- **`status`** is a Boolean, True if the task is complete and False otherwise
- **`type`** is a Boolean, True if it is a common task and False otherwise

Assumptions: The order of how you return things does not matter. In a single given task_log, there will be no repeated task names.

Return value: A list of strings; the names of tasks that match the given criteria

Example:

```
tasks = [
          ('medbay scan', True, True),
          ('wires', False, True),
          ('trash chute', True, False),
          ('asteroids', True, True),
          ('telescope', False, False)
        ]


# filter with both parameters, no ignored names
filter(tasks, True, True)            →   ['medbay scan', 'asteroids']
filter(tasks, True, False)           →   ['trash chute']
filter(tasks, False, True)           →   ['wires']
filter(tasks, False, False)          →   ['telescope']


# filter on task type
filter(tasks, task_type=False) → ['trash chute', 'telescope']


# filter on status
filter(tasks, status=True) → ['medbay scan', 'trash chute', 'asteroids']


# filter with ignored names, requires a status and a type!
filter(tasks, True, True, 'asteroids')    →   ['medbay scan']
filter(tasks, True, True, 'asteroids', 'medbay scan')   →   []
```

`check_common(task_log_A, task_log_B)` **[20pts]**

Description: During a game of Among Us, all crewmates have common tasks they must all complete. An imposter would not necessarily know what common tasks the group has to do. Write a function that compares two different player task logs to determine if two players have the same common tasks.

You must **correctly** call your filter function in this definition for full credit.

Parameters: two task logs, as described in the filter function

Assumptions: task logs are not necessarily the same length.

Return value: `True` if the players have the same common tasks regardless of status, `False` otherwise

Examples:

```
tasks1 = [ ('medbay scan', True, True),
           ('wires', False, True),
           ('trash chute', True, False),
           ('asteroids', True, True),
           ('telescope', False, False) ]

tasks2 = [ ('medbay scan', True, True),
           ('wires', False, True),
           ('trash chute', False, False),
           ('asteroids', False, True) ]

task3 = [ ('medbay scan', True, True),
          ('wires', False, False),
          ('trash chute', True, True),
          ('asteroids', True, False),
          ('telescope', False, False) ]

task4 = [ ('medbay scan', True, True),
          ('trash chute', True, False),
          ('asteroids', True, True),
          ('telescope', False, False) ]

# same common tasks in both, even though the status of 'asteroids' and 'trash chute'
# are different in tasks2, and it is missing the 'telescope' task
check_common(tasks1, tasks2)    →    True

# tasks3 has all the same task names however…
# 'wires' and 'asteroids' aren't common in tasks3, but they are in tasks1
# 'trash chute' is common is tasks3, but not common in tasks1
check_common(tasks1, tasks3)    →    False

# the ordering of the arguments shouldn't change the result!
check_common(tasks3, tasks1)    →    False

# tasks4 is completely missing a common task from tasks1 (wires)
check_common(tasks1, tasks4)    →    False
```