Assignment 4

Lists, Strings, Dictionaries, and *Impostors*

SUBMISSION REQUIREMENTS: Submit a single zip file called **assignment4.zip**. It must contain all of your question code and sample data. Information regarding deductions for late submissions, invalid submissions, and grade disputes are available on the cuLearn assignment submission page.

MARKING NOTES: This assignment has **??? marks.** A full marking scheme with notes, deductions, and policies will be made available on the cuLearn assignment submission page shortly.

Amidst Ourselves: A Game of Deception

7 Characters enter the space ship.

1 Character isn't what they seem.

There's a popular new game in COMP1405 - *Amidst Ourselves* (*definitely* no relation to the real world game, *Among Us...*). In *Amidst Ourselves*, there are seven astronauts aboard a spaceship. Each astronaut is represented by a *colour*. They run around, performing daily jobs on the spaceship. One of those colours *is not who they seem*. One of them... is an *alien*.

The crewmates want to get their tasks done. The alien wants them *gone*. Every once in a while, the players meet up and chat with each other to try and find the alien. They usually do this by talking about what they did since the last meeting. Then they vote for somebody to eject out of the airlock - hopefully saving the crew from destruction!

However; the alien will *lie*.

The main problem in finding the alien is that all of this chat can be really hard to follow. *That's where we come in.* We're going to be building a program that builds some representation of the in-game map and reads text logs from a round of the game. Using the map and our chat logs, we'll find out just who is sus(picious) and lying about where they were...

What are our goals?

Over the past few weeks, we've been learning about tools that let us build bigger, more complex, and more practical programs. Reading and writing files enables us to work on large amounts of data that we wouldn't want to enter by hand, lists let us store a variable quantity of sequential data, and dictionaries allow us to store associative data in an understandable, meaningful way.

Importantly, we can combine these tools together to form *new* tools specific to our needs. In this assignment, expect to be combining lists with dictionaries, dictionaries with dictionaries - **make sure you understand the data and the problem before trying to code!**

Please Be Aware: There is *much less* information on how to solve each question. This assignment, you are expected to spend more time planning and understanding the problem.

What if I don't like games?

I get it! We keep referencing games. What's important to see here is that the techniques you're using can be applied to lots of places. You are **not expected** to know or play the games referenced in these assignments. If a section is unclear based on a misunderstanding of what "the game" looks like, please let me know on the cuLearn forums, as you are not expected to research the context outside of this assignment specification, and I can try to clarify.

Overview

This assignment slowly evolves the functionality of our program, one step at a time.

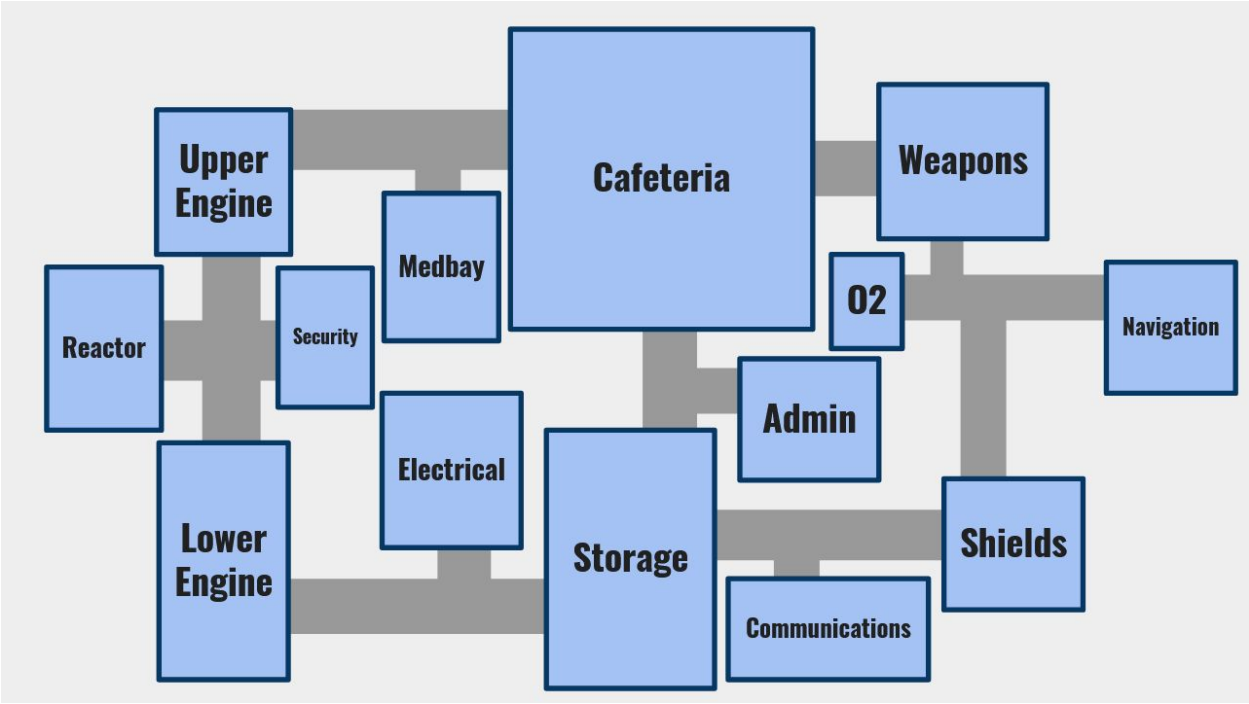
1. Read in game map data and store it in a useful way - as a dictionary
2. Write a helpful function that takes in a chat message and returns only the useful parts
3. Write a function which uses the previous helper function to reduce a full file worth of chat
4. Write a function which goes through that simplified chat log and outputs the player's votes for the round
5. Write a function which goes through that simplified chat log to check if the paths people claim to have taken are actually valid, based on the map information

Like in A3, you **must** follow naming conventions and function signatures precisely. This means using a main() function correctly, naming functions *exactly* as written in the specification, accepting the right types of inputs, and returning the correct values. You should only have print statements where explicitly mentioned in the assignment - if mentioned in the assignment at all.

Problem 1: Loading the Map

Background Information

The map of the game is represented by the image below:



The blue boxes represent each room of the space ship and the gray lines are hallways that connect the rooms. A room is said to be *adjacent* to another room if there is an uninterrupted path from that room along the gray lines without crossing another room. This is to say, rooms connected by one hallway are considered adjacent to each other.

Some examples:

- **Medbay** is adjacent to **Upper Engine** and **Cafeteria**
- **Storage** is adjacent to **Lower Engine, Electrical, Cafeteria, Admin, Communications,** and **Shields**

Our first problem is finding a way to store this map data in Python in a way that we can use for future questions. This is *associative data* - each room has a list of rooms you could move to.

More information about the question on the next page

Problem 1: Loading the Map (continued)

What are we doing?
You've been provided with a file called *skeld.txt*. This file represents every room on the map and which rooms they are adjacent to. Each line lists the room name and a comma separated list of rooms that it connects to.

Be aware that we should be able to load a **different map** with **different room names** and it will work just fine.

Write a function called **load_map(file_path)**.

- The function **takes in a string** representing the filepath of the map file
- You will **create and return** a dictionary which represents the map
    - The *purpose* of this dictionary is to let us look up a room and see which rooms we could move to
    - The *keys* of the dictionary are the name of the room (string)
    - The *values* are a **list of strings** representing every room which is adjacent to the room in the key
- To test it, you can write some code in your main() function
    - This is optional, but obviously highly recommended to make sure it works
    - Store the return value of **load_map("skeld.txt")** in a variable in your main() function and print it out to see if it matches what you think it should.
- **Tip:** If dictionaries are still confusing, start by trying to hard code a small version of the dictionary in Python (two or three rooms) by hand to get a better idea of what you'll be constructing when you read the file!
- **Tip:** One entry in your dictionary will be something like…
    - "room name": ["another room", "room", "some other room"]

Problem 2: Simplifying Messages in Chat

Background Information

Every once in a while, our players will meet up to discuss what everyone has been doing - usually because our alien has snacked on one of the crewmates.

The crew is expected to perform three basic tasks here:

- Talk about where they were throughout the round
- Talk about who they've seen throughout the round
- Vote for somebody they want to shoot out of the airlock… *Presumably* because they think they are an alien.

The conversation can be difficult to follow, but this is your main place to find the alien. People are expected to state where they were, and what rooms they passed through. If they claimed to go to a room they couldn't have reached, later on in the assignment we'll identify them as a liar.

> **Side Note: But this isn't accurate!!!**
>
> ---
>
> If you've played the game *Among Us*, you may notice some parts of this assignment are... *off*. The program you are building for this assignment is not intended to work with all chat logs, under all circumstances, nor is it meant to improve your win rate.
>
> To build something that can accurately parse fully unstructured conversation is a much more complex task - but one I encourage you to try out after completing the assignment! How would you make a "lie detector" for real world game data? What challenges might you face?

For the purposes of this assignment, assume that everyone is *shockingly* consistent in how they talk.

- Assume crewmates can only have the following colours:
  - ["red", "blue", "green", "yellow", "brown", "pink", "orange"]
- If they do not mention a location name, assume it is useless information
- If they mention a location name and no colours, assume they are talking about themselves
- If they mention a location and a colour, assume they are claiming another crewmate was in a location
- **Nobody will <u>ever</u> say the word "vote" or "voted"**... for some reason.

More information about the question on the next page

Problem 2: Simplifying Messages in Chat (continued)

What are we doing?
In this question, we want to take complex chat messages and reduce them to only their important parts.

- We want to know who is speaking; what colour are they?
- We want to know who they are talking about; are they talking about themselves, or someone else?
- We want to know where they are talking about; what room were they in?

You will be writing a function called **simplify_testimony(chat, rooms)** which:

- Takes in a chat message as a string
    - Messages are either of the form "**speaker_colour: this is the message"**
    - ...Or the form "**colour voted colour"**
    - Assume each chat message will only ever refer to at most one colour and one location at a time
- Takes in the spaceship rooms dictionary that you built in Problem 1
- It returns a string, which is a simplified version of the message of the form:
    - **"*speaker*: *colour* in *room*"**, explaining who was in which room, and who made the claim
    - If no room was mentioned, return an empty string, ""
    - If it's a vote, simply return the vote line without any newline characters.
- **Tip:** Start by recognizing the different cases you will need to account for
- **Tip:** Whenever I do text processing, I find it helpful to play around in the Python interpreter with some sample strings to try and hone in on how the text breaks apart in different situations.
- **Tip:** Nobody will *ever* say the words "vote" or "voted". This means the only time they will show up is in votes.
- **Tip:** It is highly likely, but not required, you will need to use these tools:
    - strip()
    - split()
    - find()
    - format()
- **Tip:** You can store the list of possible colours as a global constant; we will assume these colours will be in every game and will never change

Example test cases for this function are on the next page

Problem 2: Simplifying Messages in Chat (test cases)

Examples (valid rooms will contain many more rooms than shown)

chat_message: "red: I ran out of cafeteria"
valid_rooms: ["cafeteria", "storage", "admin", "medbay"]
Actual: `simplify_testimony(chat_message, valid_rooms)`
Expected: "`red: red in cafeteria`"

chat_message: "red: On my way, I saw blue in admin."
valid_rooms: ["cafeteria", "storage", "admin", "medbay"]
Actual: `simplify_testimony(chat_message, valid_rooms)`
Expected: "`red: blue in admin`"

chat_message: "orange: Where were you after that?"
valid_rooms: ["cafeteria", "storage", "admin", "medbay"]
Actual: `simplify_testimony(chat_message, valid_rooms)`
Expected: ""

chat_message: "green voted orange"
valid_rooms: ["cafeteria", "storage", "admin", "medbay"]
Actual: `simplify_testimony(chat_message, valid_rooms)`
Expected: "`green voted orange`"

Problem 3: Loading Chat from a File

Simplifying individual messages is fine, but really we want to use that function to process *a bunch* of chat messages at one time.

What are we doing?

For this problem, you will write a function called **load_chat_log(filename, rooms)** which:

- Takes in the filename of a chatlog; we'll provide you with a sample called *chatlog.txt*
- Takes in the rooms dictionary that you built in Problem 1
- It will read the entire chat log and simplify each line using the **simplify_testimony(...)** function you wrote in Problem 2
- It will return a list of strings representing the simplified chat log
- We'll ignore any invalid messages (empty strings)
- For example: If the above four chat messages were in a file, we'd expect **load_chat_log(...)** to return:
    - ["red: red in cafeteria", "red: blue in admin", "green voted orange"]

Problem 4: Tallying Votes

Background Information

Once we have all of our chat information collected and simplified, we want to actually *do* something with it. A critical piece of COMP1405's favourite game, *Amidst Ourselves*, is that players will **vote** for who they think should be thrown out of the airlock. This usually takes the form of a message in the chat log saying something like, "**blue voted orange"**. They may also vote to **skip** if they don't think there's enough information to eject someone to their doom. This looks like, **"orange voted skip"**

What are we doing?

Write a function called **tally_votes(chat_log)** which:

- Takes in the simplified chatlog, which is returned from **load_chat_log(...)** written in Problem 3; this should be a list of simplified strings
- Create a dictionary to track how many votes each character has, and how many votes there are for skipping
    - This dictionary should include "zero" votes for people who were not voted for, and still include "skip" even if nobody voted skip
- Return the dictionary with the voter information
- Remember: The crewmates can be these colours:
    - ["red", "blue", "green", "yellow", "brown", "pink", "orange"]
- **Example:**
    - Given the chat log:
        - red: I saw green in communications, acting pretty sus.
        - **orange voted green**
        - red: Then I went up to navigation and that's where I found the body.
        - **green voted orange**
        - blue: I can also confirm that green was in communication.
        - orange: Where were you after that?
        - blue: I went straight to upper engine.
        - **blue voted skip**
        - **red voted skip**
    - We expect a dictionary that resembles:
        - {"red": 0, "blue": 0, "green": 1, "yellow": 0, "brown": 0, "pink": 0, "orange": 1, "skip": 2}
- **Tip:** You will need to find a way to differentiate a chat line from votes
- **Testing Tip:** As usual, I suggest you try to test all of this in your main function any way you see fit! You can even create multiple files to test different things at once.

Problem 5: Identifying Paths

Background Information
When crewmates are talking about what they did in a round, they usually say that they went from one location to another, to another, to another. This path information is important, as the alien has the ability to jump around the map and skip over rooms!

What we want to do is read through our chat data and see if we can list out the paths everyone is saying they took. Some people might not speak up. Some people might call out where *other* people were, instead of themselves. We don't care about this! We only care about where people said *they were themselves.*

What are we doing?
In this problem, you'll be writing a function which reads our simplified chat data and returns a dictionary containing the path that each crewmate claims to have taken.

You will write a function named **get_paths(chat_log)** which:

- Takes in the simplified chat log from Problem 3
- Returns a dictionary with all of the path information
    - The keys should be the colours of each crewmate; even if they didn't mention their path data
    - The values for each colour will be a list
        - The list will be a list of strings
        - Each string is a room name
        - The list should be ordered with 0 being the first room they entered, and then the next room, and the next room…
    - If a crewmate did not mention their path, the value should be an empty list.
- **Tip:** You will need to find a way to differentiate chat lines from votes
- **Tip:** A crewmate will **only** mention another colour if they are calling out **that colour's** path. Ex. **red: blue was in reactor**, calls out blue, and shouldn't be considered in anyone's path. We **only** care about someone talking about themselves

Problem 6: Who is Lying?

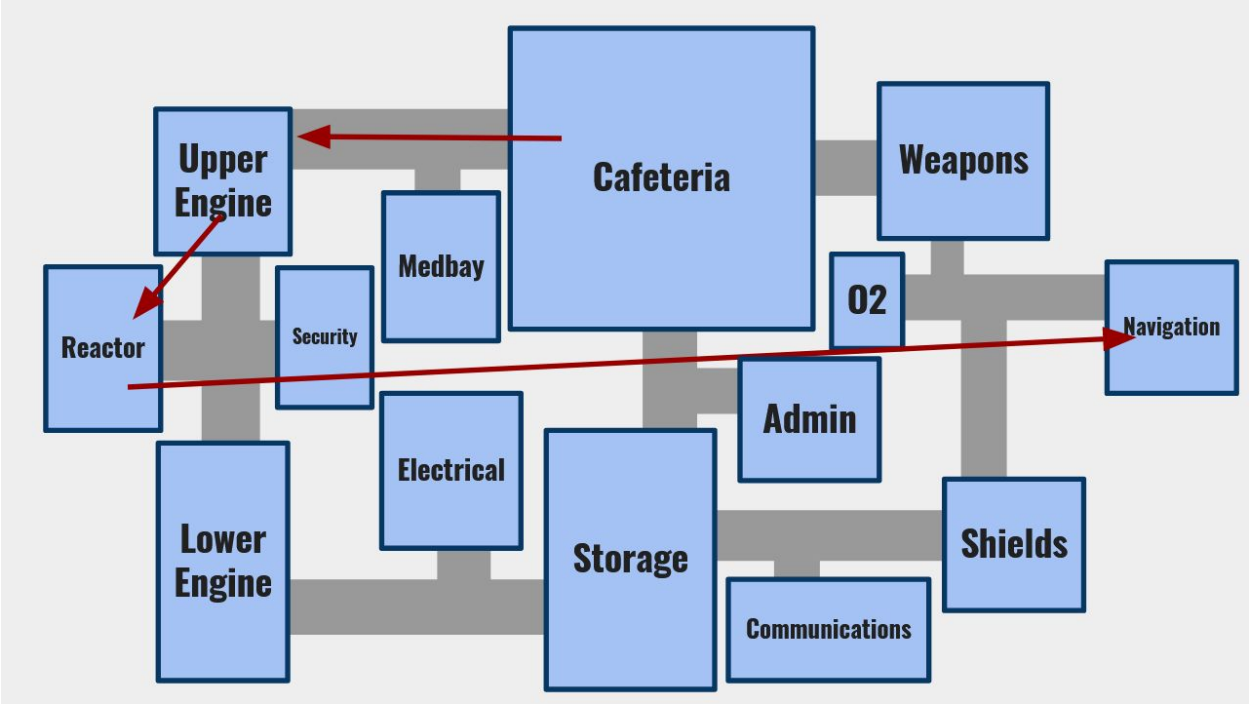
Background Information

The alien is motivated to *lie* about where they were. Unfortunately for them, it can be really hard to lie about what path you took! What we want to do is look at all of our paths from Problem 5 and compare it to the map that we built in Problem 1.

Did **red** try to say they went from one room to a different one, that isn't connected to it? For example:

- red: So I was hanging around **cafeteria**
- red: then I moved over to **upper engine**
- red: I saw blue in security
- red: I did a task in **reactor**
- red: and then I went straight to **navigation**

On our map, that path looks like:



Cafeteria to Upper Engine to Reactor is a fine path, but navigation isn't connected to reactor! How in space did they get there? That's a **suspicious (or *sus*) path**.

More information on next page

Problem 6: Who is Lying? (continued)

What are we doing?

Luckily for us, we've already built functions that will help here: our **load_map(...)** provides us with a dictionary containing which rooms are connected to which, and **get_paths(...)** tells us where people claimed to be walking around. We want to write a function which takes in that data and returns a list of every colour that appears to have lied about where they went.

Write a function called **get_sus_paths(path_dict, rooms)** which:

- Takes in the dictionary of paths that were returned by our **get_paths(...)** function, telling us where everyone went
- Takes in the dictionary of rooms that were returned by **load_map(...)**, telling us which rooms are connected to each other
- Returns a list of strings, where each string is the colour of suspicious crewmates
- A crewmate is considered suspicious if at some point they claimed to have moved from one room to the next - but that room was **not attached to the room they came from**.
- **Tip:** It may help to draw out your dictionaries and walk through the problem.
- **Tip:** There are many ways to do this, but it can be accomplished in as little as 10 lines of code; don't feel obligated to keep it small, but recognize when you're going *way* too long and there's likely a simpler solution.
- **Tip:** You will almost certainly need to go through the chatlog data by hand to see which characters lied - **before** you start coding.

Final Notes

You are not asked to print anything throughout this code - this doesn't mean you absolutely shouldn't. You shouldn't print anything out if we didn't tell you to *within* the functions, but please feel free to print data and test things out in your main() function!

If you are struggling with problem solving, that's okay. You are likely to spend most of your time on this assignment trying to work out the problem. As we go further in the assignments, you will be provided with less guidance. This is to say, just because you don't get it at first - keep trying! Look at it from different angles. Look at the problem solving steps from Lectures 14 and 15 and see if you can solve it in that way.

To print dictionaries and lists nicer for testing, you can use the module **pprint** which is included in Python. You can see an example of that in Lecture 14. You can import pprint, like: **`from pprint import pprint`** and then use pprint() as if it were a regular print statement (for a single value), like **`pprint(my_dictionary).`** This isn't required, but I find it helps me!

Updates to this assignment may come out over time, if there are issues with the data that weren't noticed at first. **Pay attention** to course announcements and the assignment clarifications forums to keep up to date on info!

In addition to the sample data: I will provide what each function should return for that data. As with assignment 3, you are expected to either modify this data or build your own in order to test different situations. This is for reference only. It is acceptable to submit just the sample data we sent. You **do not** need to output the data to a file, or resubmit the example output - **this is for your reference only**.

Recap

- Your cuLearn submission should be a single file, **assignment4.zip**
- Your zip file should contain **one Python file, assignment4.py**
- Your zip file should contain the sample data used to test your program
- Late submissions will receive a 2.5%/hour deduction up to an 8 hour cut-off period
- Invalid submissions (incorrect name, incorrect function names) will receive a 10% deduction immediately
- As usual, you are expected to submit periodically; as you complete questions, try to submit to cuLearn, just in case of data loss or last-minute submission problems.
- It is your responsibility to verify the submitted files work correctly - redownload and try them again