Ecological Survey Tool Assignment

Learning objectives

This laboratory is intended to give you practice with a few core skills:

- Reading data from plain-text files and representing them using Python dictionaries and sets.
- Using **sets** to compute intersection/union/difference efficiently.
- Building a small **console** interface that parses user input and handles user error without crashing.
- Using conditionals to redirect flow to the right function calls with the correct parameters.
- Writing and using a **generator** function.

1 Introduction

In this lab, you will build a simple, console-based application that assists biologists in understanding the data collected during their ecological explorations. In each expedition, biologists visit a site and write down all of the species of animals that they see at that site. These lists are saved as .txt files, where the first line contains the name of the site, and all of the subsequent lines contain a species spotted at that site. Because these lists are created by humans and we can be forgetful, it's possible that species will appear multiple times in each file.

2 Data format

Each expedition produces a single `.txt` file with the following format:

- **Line 1**: *Name of site.*
- **Lines 2+**: one *species name* per line.
- Blank lines may appear and should be ignored.
- Duplicate species may appear.
- All matching and comparisons are **case-sensitive**.

Example files can be found in the Example scenario.

3 Example scenario

Say four biologists took four different expeditions:

- Rita went to a `cave`
- Lunara went to a `dense_forest`
- Claudia went to a `dense_forest`
- Daniel went to a `volcano`

They produced the following text files:

cave_rita.txt

```
cave
pit viper
bat
leafcutter ant
pit viper
tarantula
tadpole
```

dense_forest_lunara.txt

```
dense forest
howler monkey
sloth
chipmunk
toucan
leafcutter ant
sloth
tarantula
pit viper
```

dense_forest_claudia.txt

```
dense forest
sloth
toucan
leafcutter ant
tarantula
white-faced capuchin
```

volcano_daniel.txt

```
volcano
tarantula
parrot
pit viper
leafcutter ant
tarantula
grasshopper
```

4 Software requirements

One text file is created for each site expedition. The first line of the text file contains the name of the site, and each subsequent line contains a species spotted at that site. The software should read the contents of these text files and store the site and species data in the appropriate data structures. After `ecological_survey.py` is run, the user should be presented with the following message in the console.

```
What would you like to do? Enter one of the following commands:

 - sites
   Show available sites

 - common <site1>, <site2>, ..., <siteN>
   Show species found in all of the listed sites.
   If no sites are given, shows species common to all sites.

 - combined <site1>, <site2>, ..., <siteN>
   Show species found in any of the listed sites.
   If no sites are given, shows all species.

 - unique <site>
   Show species found only in the specified site (not in others).

 - species_in <species>
   Show sites that species is found in.

 - q or quit
   Exit the program.

 - h or help
   Show these instructions again.
```

The user should then be able to enter any of the commands into the console and get the expected response. Read on for more information on these commands.

4.1 sites command

The `sites` command shows all sites that are available. In the example scenario, these are `dense_forest`, `cave`, and `volcano`.

Expected console behavior:

```
Enter command: sites
dense forest
volcano
cave
Enter command:
```

4.2 common command

The `common` command can take any number of sites as arguments. It shows the species that are common to all of the sites provided. If no arguments are provided, it should show species that are common to all sites.

Expected console behavior for the example scenario:

```
Enter command: common cave, dense forest
pit viper
tarantula
leafcutter ant
Enter command:
```

4.3 combined command

The `combined` command can take any number of sites as arguments. It shows the species found in any of the sites provided, i.e., the union of the sets of species for each site provided. If no arguments are provided, it should show all species discovered across all expeditions.

Expected console behavior for the example scenario:

```
Enter command: combined volcano, cave
leafcutter ant
bat
pit viper
tadpole
tarantula
grasshopper
parrot
Enter command:
```

4.4 unique command

The `unique` command takes one site as an argument. It shows the species that are unique to that site, i.e., that can only be found at that site.

Expected console behavior for the example scenario:

```
Enter command: unique cave
tadpole
bat
Enter command:
```

4.5 species_in command

The `species_in` command takes one species as an argument. It shows all of the sites where this species has been found.

Expected console behavior for the example scenario:

```
Enter command: species_in leafcutter ant
dense forest
volcano
cave
Enter command:
```

4.6 quit command

The `quit` command shows a goodbye message to the user and ends the program. It can be executed by entering `quit` or just `q`.

Expected console behavior:

```
Enter command: q
Goodbye.
```

4.7 help command

The `help` command shows information on the available commands to the user. This should be the same set of instructions shown when the user runs the program. It can be executed by entering `help` or just `h`.

Expected console behavior:

```
Enter command: help

What would you like to do? Enter one of the following commands:

 - sites
   Show available sites

 - common <site1>, <site2>, ..., <siteN>
   Show species found in all of the listed sites.
   If no sites are given, shows species common to all sites.

 - combined <site1>, <site2>, ..., <siteN>
   Show species found in any of the listed sites.
```

```
   If no sites are given, shows all species.

 - unique <site>
   Show species found only in the specified site (not in others).

 - species_in <species>
   Show sites that species is found in.

 - q or quit
   Exit the program.

 - h or help
   Show these instructions again.
```

4.8 User error handling

When the user makes a mistake, the program should not crash. Instead, it should provide the user with a user-friendly error message that allows them to try again. Verify that you account for the possibility of typos in commands or in arguments provided. Here are some examples (your messages don't have to match mine):

```
Enter command: uniqu cave
That was not recognized as a command. Please try again or enter 'help' for
more support.
Enter command: unique
The ‘unique‘ command expected one argument. Received: 0. Enter 'help' for
more information.
Enter command: unique cave
bat
tadpole
Enter command:
```

```
Enter command: common cave, dense forets
At least one of the provided arguments was invalid: ['cave', 'dense forets']
Enter command: common cave, dense forest
leafcutter ant
tarantula
pit viper
Enter command:
```

5 Submission requirements

For a program of this scale and larger, it is important to break up the task into modular components (functions). While there are many ways to make this software, please begin with the starter code provided and fill in the functions in the following order. Feel free to create helper functions along the way. Once you finish, test your software, making sure that

it is behaving as expected. Note that the software should be runnable by entering `python3 ecological_survey.py` in the console. Then, at the top of the Python file as a comment, write a reflection on why you think sets are more appropriate for representing species spotted at sites than lists would be.

5.1 process_site_files(file_path)

This function should take a list of file paths (strings) as an argument. For each file path, it should read the .txt file, extract the site name, species spotted, and return a dictionary where the keys are the sites, and the values are sets of species.

5.2 get_arguments(user_input)

This function should take a string as input which represents one or multiple arguments that a user provided, separated by commas. The function should separate the arguments and return a list of strings, where each item in the list is a different argument. For example, if user_input = " cave, dense_forest, volcano", it should return the list ["cave", "dense_forest", "volcano"].

5.3 species_in(species, sites_species_dict)

This generator function should take a species and a dictionary mapping sites to sets of species as arguments. It should then *yield* sites where the species has been spotted.

5.4 unique(site_species, *other_site_species)

This function should take `site_species`, a set of species found in a site of interest, and any number of `other_site_species` arguments which represent sets of species from other sites. It should return a set of species that are only found in `site_species` and not in any of the `other_site_species`.

5.5 common(*species_sets)

This function takes any number of sets of species. It should return a set containing only the species that are found in all of the sets, i.e., the intersection of all sets.

5.6 combined(*species_sets)

This function takes any number of sets of species. It should return a set containing all species found across all sets, i.e., the union of all sets.

5.7 run_ecological_survey()

This function is responsible for processing the expedition text files and representing that data in the appropriate Python data structures. Use the pre-written `get_expedition_files()` function to get a list of file paths for the expedition files. Then use the appropriate function that you wrote to process the data into a data structure.

Then, provide the user with instructions on what commands are available to them, and handle user commands so that the software makes the appropriate responses. Closely reference the software requirements section and the expected console behavior examples while working on this.