Lab 5: Testing, Exceptions and File Handling

Rahul Kukreja & Jeffrey Yim

jyim3@bcit.ca

Welcome!

In today's lab, you will:

1. Implement a simple dictionary program that reads and writes data to a file.
2. Demonstrate proficiency writing code that is efficient, easy to understand, and correct.
3. Identify crucial functions and write meaningful unit tests
4. Continue to bask in the joy of programming. Because you do love it, don't you!

Grading

This lab and all future labs will be marked out of 10

For full marks this week, you must:

1. (2 point) Correctly upload your code to Github so I can grade your solution
2. (6 points) Generate a correct solution to the problem(s) in this lab
3. (2 point) Follow PEP-8 standards to correctly format and comment your code. This also includes best practices such as meaningful identifier names, breaking down code into re-usable functions, etc.

Requirements

Please complete the following:

1. Clone the project from: **https://classroom.github.com/a/agoig44c**

1. There is a JSON file included with the project that contains (perhaps) all the words in the english dictionary. If not all, then definitely a lot. Each word may have more than one definition associated with it. Open up the file in a text editor and take a look at the data. Does this remind you of a container data structure?
2. This lab onwards I'm going to be picky about how your modules are arranged. So, just for this lab I'm going to specify the module structure:

    - `driver.py` - This module is responsible for housing the dictionary class (alongside any custom exceptions that apply to the dictionary) and any driver methods that run the program.
    - `file_handler.py` - This module is responsible for housing any classes and/or structures that are responsible for reading and writing to files. This includes any custom exceptions.
3. You are going to write a dictionary program that is going to demonstrate the following:

    1. Loading definitions from a JSON file.
    2. Querying the user for an input (a word) and returning any definitions that match with the given input. Your program should not be case sensitive. That is, `Rain`, `RAin`, and `rain` should all return the same definition.
    3. The program should save all the words and definitions queried to a text file.
    4. Your ability to handle any errors and edge cases gracefully.
2. In true Object Oriented fashion, we are going to encapsulate all file behaviors in a separate entity. Your program should support 2 file types: `.txt` and `.json` . A good way to implement this is to create an `enum` type. You can look at the python documentation for enums to see how they are used and implemented. (https://docs.python.org/3/library/enum.html)
3. Implement a `FileExtensions` Enum that has 2 valid states, `TXT` and `JSON`.
4. Implement a `FileHandler` class that has the following methods and features:

    - A `load_data(path, file_extension)` static method. This method is responsible for checking the file extension and reading the file accordingly. This method is also responsible for raising any exceptions if they occur. You might need to look up the `pathlib` module to be able to check if the file exists.
    - A `write_lines(path, lines)` static method. This method is responsible for writing the given lines to a **Text** file. The text file should be appended to and not overwritten.
    - Anything else you deem relevant and necessary.
2. The `file_handler.py` should also house an `InvalidFileTypeError` class. This exception should be raised if the user attempts to read a file that is not a `.json` or a `.txt` file (or in other words, if the user attempts to read a file that is not supported by the `FileExtensions` enum).
3. Your dictionary should implement the following methods and features:
    - A dictionary needs to be loaded. Implement a `load_dictionary(self, filepath)` method that is responsible for loading data into a dictionary
    - A dictionary needs to be able to look up the definition of a word. Implement a `query_definition(self, word)` method that returns the definition(s) of a given word.
    - A way to keep track of whether the dictionary has been loaded or not. This will be useful when testing your applications.

Optional: Use the `difflib` library to suggest similar words in case the user enters an incorrect word and it isn't found in the dictionary. Remember, the program should not be case sensitive! `RAInn` should return `rain` as a suggestion.

4. Remember to raise and handle any exceptions appropriately. Declare custom exceptions if you think it is necessary.
5. Write meaningful unit tests for your program. Make sure you write unit tests that test with both correct and incorrect input. Ensure you test the following:
    - Loading data into a dictionary.
    - Writing data to a text file. What assertion can you use to test this?
    - Querying a word.
    - Anything else you deem necessary
6. Your program should load the dictionary and provide the user with the option to keep querying definitions until the user has entered `exitprogram` as their input. Remember to format your output and any definitions nicely.
7. Don't forget, your program should use the `try-except-else-finally` or any variation of it to handle any errors that your code might encounter. Where and when you raise and handle errors will depend on the architecture of your code and how your functions are logically laid out.
8. Also don't forget to save the definitions and words that the user queries to a text file.

Ensure you push your work to github classroom. I'd like to see sensible git commits comments, and commits must take place at logical points in development.

That’s it. Good luck, and have fun!