Name: Silvano Ross
Date: March 4, 2022
Course: IT FDN 110A
GitHub: https://github.com/silvanoross/IntroToProg-Python

Assignment 07 - The Pickling Exception

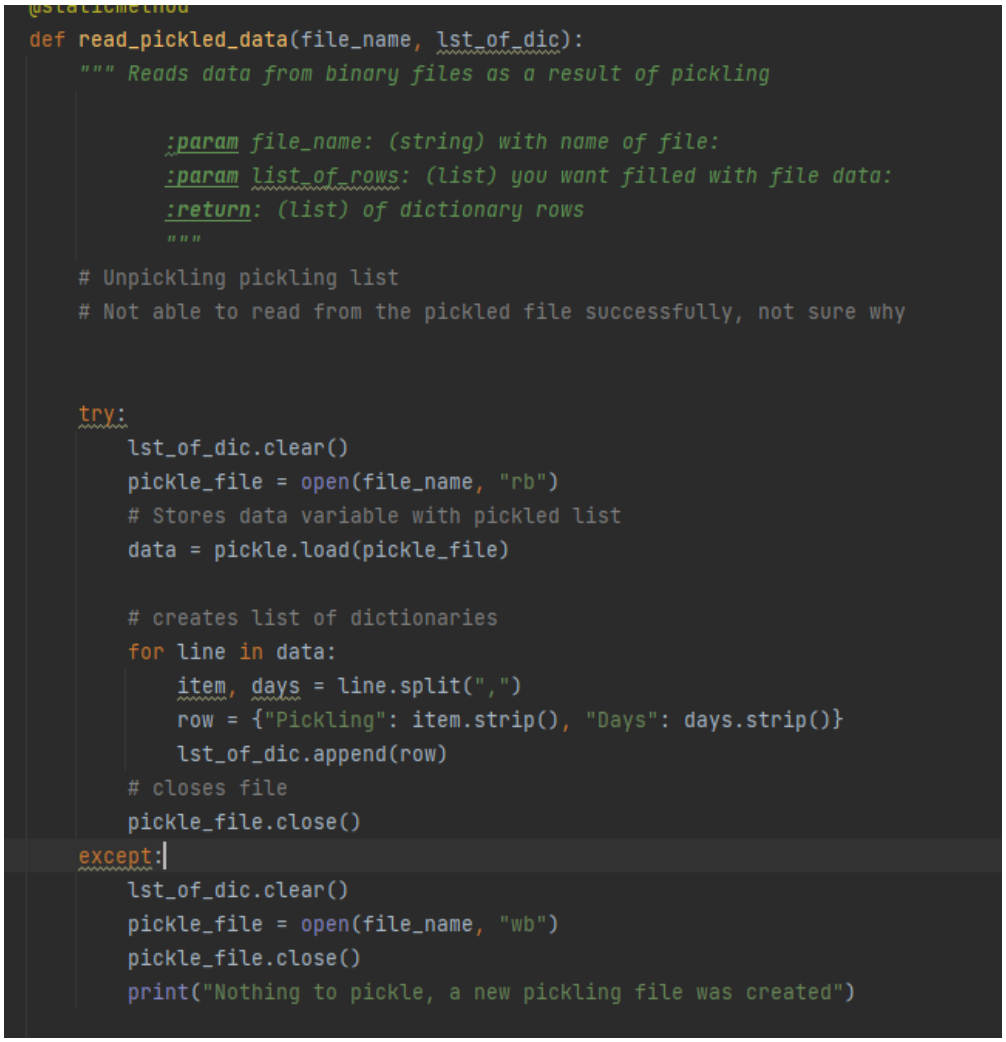
INTRODUCTION

This week we are officially on our own. Our job is to demonstrate how pickling and structured error-handling work in python. So far we have used relatively in-efficient forms of storing and accessing data. Through pickling data we are better able to utilize a computer's memory when reading and writing data. Further, we have only been relying on the built-in error handling that comes with python. Through structured error handling we can create more custom notes for specific kinds of errors to better explain to a user what may have gone wrong.

PICKLING - Just Like a Jar of Pickles

The term pickling in python holds a meaning that is in fact quite close to the same term used in food storage. Much like how one pickles items to preserve them for later use (also somewhat altering the shape and flavor of the item), pickling in python refers to storing data in a different form from its original for later use. Pickling requires that information be stored in binary files rather than text files or other formats. This creates files that better utilize a computer's working memory.

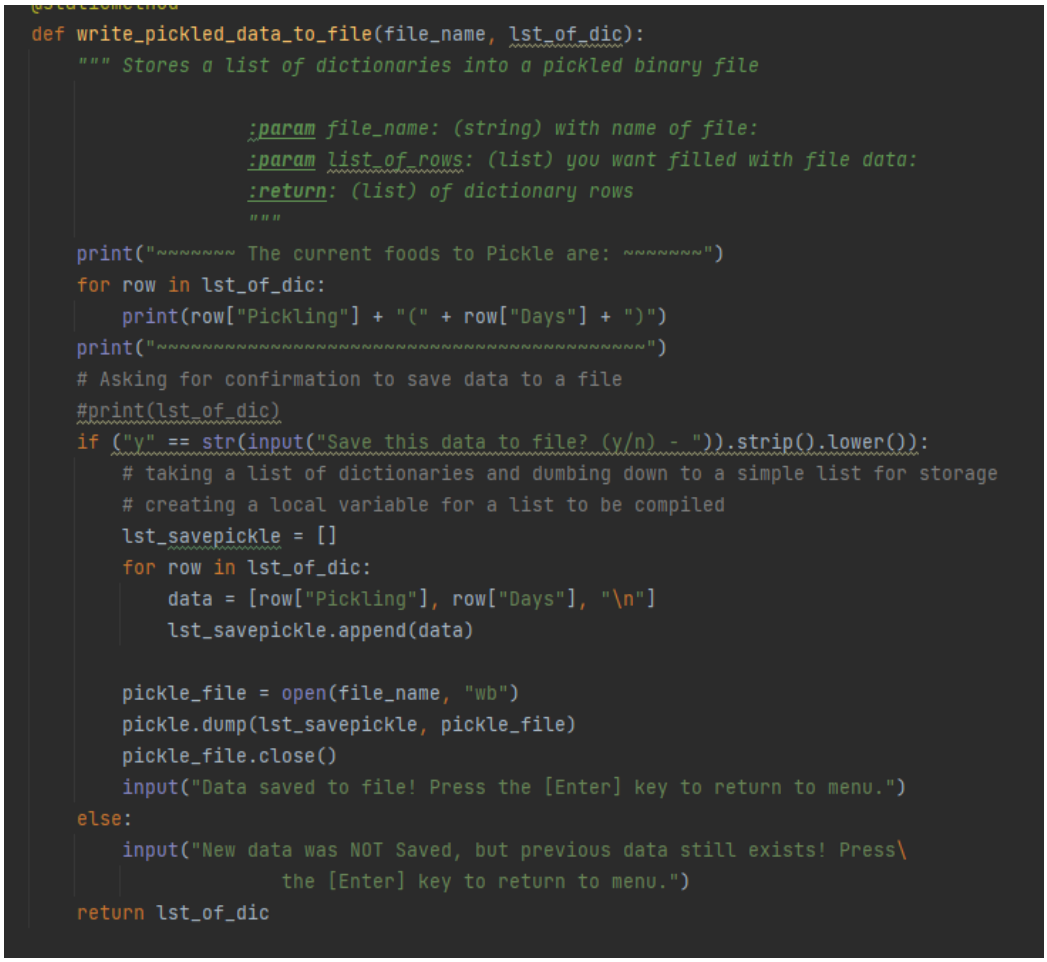
This week I decided to turn the code from Assignment 06 that wrote text files into code that did something similar, but wrote pickled binary files instead. This involved a big work-around of many of the variable names and a few of the user-defined functions. Ultimately I was able to successfully re-create the program to write user-inputted data to a binary file. The only trouble I had was getting the program to read from a binary file and store the data into a successful working format.



```
@staticmethod
def read_pickled_data(file_name, lst_of_dic):
    """ Reads data from binary files as a result of pickling

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
    # Unpickling pickling list
    # Not able to read from the pickled file successfully, not sure why


    try:
        lst_of_dic.clear()
        pickle_file = open(file_name, "rb")
        # Stores data variable with pickled list
        data = pickle.load(pickle_file)

        # creates list of dictionaries
        for line in data:
            item, days = line.split(",")
            row = {"Pickling": item.strip(), "Days": days.strip()}
            lst_of_dic.append(row)
        # closes file
        pickle_file.close()
    except:
        lst_of_dic.clear()
        pickle_file = open(file_name, "wb")
        pickle_file.close()
        print("Nothing to pickle, a new pickling file was created")
```

Figure 1.1 Taken from assignment 07

I had originally stored the pickled data as a simple list:



```
def write_pickled_data_to_file(file_name, lst_of_dic):
    """ Stores a list of dictionaries into a pickled binary file

                :param file_name: (string) with name of file:
                :param list_of_rows: (list) you want filled with file data:
                :return: (list) of dictionary rows
                """
    print("~~~~~~~~ The current foods to Pickle are: ~~~~~~~~")
    for row in lst_of_dic:
        print(row["Pickling"] + "(" + row["Days"] + ")")
    print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
    # Asking for confirmation to save data to a file
    #print(lst_of_dic)
    if ("y" == str(input("Save this data to file? (y/n) - ")).strip().lower()):
        # taking a list of dictionaries and dumbing down to a simple list for storage
        # creating a local variable for a list to be compiled
        lst_savepickle = []
        for row in lst_of_dic:
            data = [row["Pickling"], row["Days"], "\n"]
            lst_savepickle.append(data)

        pickle_file = open(file_name, "wb")
        pickle.dump(lst_savepickle, pickle_file)
        pickle_file.close()
        input("Data saved to file! Press the [Enter] key to return to menu.")
    else:
        input("New data was NOT Saved, but previous data still exists! Press\
                    the [Enter] key to return to menu.")
    return lst_of_dic
```

Figure 1.2 Taken from Assignment 07

I figured by storing the pickled data as a simple list (lst_savepickle) from its original dictionary list it would make for easier reading and writing from a binary file. This however was not the case and the program continues to make a new binary file with each run through. I was not able to fix this bug.

STRUCTURED ERROR HANDLING

The only instance of structured error handling I incorporated into this program was with the user input. I made it so the 'item' variable within the input() function had to be a string variable and the 'days' variable had to be an integer or a floating point value.

```
while (True):
    # Step 3 Show current data
    IO.output_current_pickling_items(lst_of_dic=lst_pickle)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    try:
        str_choice = str(IO.input_menu_choice())  # Get menu option
        if int(str_choice) <= 4:
            print("you chose option", str_choice)
            pass
    except (ValueError, TypeError):
        print("Make sure to enter a numerical option 1-4")
        pass
```

Figure 2.1 Taken from assignment 07

For the menu option I stopped the program from crashing if the user were to enter a non integer or if the integer was out of the range of what was available on the menu of options. This was done with a try, except block with a diminutive of structured error handling.

SUMMARY

This week we learned how to pickle data into a binary file. This helps with more efficient memory usage for computers. Files can be written and accessed in a much different way than when working with a text file. Pickling allows the user to store any kind of data whether it be a dictionary, tuple, list etc. This week we also went through structured error-handling. Having fail safes built into your program can accommodate actions that would otherwise result in unexpected crashes.

Bibliography

1. https://www.geeksforgeeks.org/understanding-python-pickling-example/
2. https://www.journaldev.com/15638/python-pickle-example#:~:text=Python%20Pickle%20load,%2Dbinary%20(rb)%20mode.