Mike Hamilton

Foundations of Programming: Python

Assignment 7

Repo: https://github.com/mah765/IntroToProg_Assignment07

11/29/21

Pickling and Structured Error Handling:

Extending the To-Do List Script

Summary

- In this assignment, we were required to research two topics (pickling and structured error handling) and then create a script that demonstrates our working knowledge of both topics.
- I decided to continue using my existing "to do list" tracker script, but add functionality for reading and writing to pickle files, and to enhance the flow of the script by using a number of try…except blocks.
- As before, the script initially reads in any existing to-do items from a file, and then asks the user to display, add, remove, or write the new results back to that same file. The script now includes two write options – one for writing to a text file and one for writing to a pickle file. The read function has been modified to automatically detect the input file type (text or pickle) and adjust accordingly.
- We were required to run this script in both PyCharm and in a command window and show a screenshot of the output. We were also required to post the files to a new Github repository.
- In the sections below, I provide more detail on this process, and provide screenshots to document key steps and outputs.

Part 1: Creating the Script

In PyCharm, I created a new project in the 'Assignment_07" directory and then opened a new Python script file. I then pasted in my syntax from Assignment 06 and made some key enhancements:

- First, I modified the existing read/write data functions to include the option of reading from a pickle file and writing to a pickle file. The read data function will automatically determine what type of file is being read in by looking at the file suffix. If it is a pickle file, it will read it in using the pickle.load function (in a loop). Otherwise it will continue to read in the data as we did in the last assignment.
- Secondly, I added a number of try…except blocks to let the script run more smoothly, providing useful information to the user rather than exiting from the script entirely. These blocks allow me to print error messages to the user to let them modify their existing inputs, or to let them know (for instance) that a file could not be found.

The script looks like this:

```
# ------------------------------------------------------------------------ #
# Title: Assignment 07
# Description: This is an enhanced version of assignment 6, with added features
#              for handling exceptions as well as for pickling the results.
#              When the program starts, we load each "row" of data
#              from an external pickle file "ToDoList.txt" into a python Dictionary.
#              We then add each dictionary "row" to a python list "table".
#              Users can choose to display the current list, add or remove items,
```

```
#              or else write the results back to file (in a pickle format).
# ChangeLog (Who,When,What):
# MHamilton,11.25.21,Created script based on assignment 6.
# MHamilton,11.27.21,Updated syntax to include error handling.
# MHamilton,11.29.21,Updated syntax to read in and write out pickle files.
# ------------------------------------------------------------------------- #

# Imports ----------------------------------------------------------------- #
import pickle           # Needed for pickling

# Data -------------------------------------------------------------------- #
# Declare variables and constants
file_name_str = "ToDoList.txt"          # Default name of the data file
file_name_str_p = "ToDoList.pickle"     # Default name of the pickled data file
row_dic = {}            # A row of data separated into elements of a dictionary {Task,Priority}
table_lst = []          # A list that acts as a 'table' of rows
choice_str = ""         # Captures the user option selection


# Processing  ------------------------------------------------------------ #
class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def read_data_from_file(list_of_rows):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        # Allow user to define file. Note we do not need a try...except block
        # here because it is taken care of in the following code chunks.
        file_name = input("Please enter name of existing file to read in:")

        # Must determine if file is text file or pickle to read in appropriately.
        # Here we extract the file suffix.
        filetype = file_name.split(".", 1)[1]

        # If file is a text file, execute this block:
        try:
            if filetype == 'txt':
                list_of_rows.clear()  # clear current data
                try:
                    file = open(file_name, "r")
                    for line in file:
                        task, priority = line.split(",")
                        # Here we make sure to convert all text to lower case to match with later
                        # functionality.
                        row_dic = {"Task": task.strip().lower(), "Priority": priority.strip().lower()}
                        list_of_rows.append(row_dic)
                    file.close()
                    return list_of_rows
                except:
                    print("File not found!")

            # Conversely if file is a pickle, execute this block:
            elif filetype == 'pickle':
                list_of_rows.clear()  # clear current data
                try:
                    with open(file_name, "rb") as file:
                        while True: # This line is needed because we don't know how many rows are in the pickle.
                            try:
                                line = pickle.load(file)
                                task, priority = line.split(",")
                                # Here we make sure to convert all text to lower case to match with later
                                # functionality.
                                row_dic = {"Task": task.strip().lower(), "Priority": priority.strip().lower()}
                                list_of_rows.append(row_dic)
                            except EOFError:      # End of data error -- indicates no more rows to be read in.
                                break
                    return list_of_rows
                except:                           # General error -- file can't be located.
                    print("File not found!")

        except:
            print("File not found or invalid format. You may add tasks and a new file will be created.")

    @staticmethod
    def read_data_from_pickle(file_name, list_of_rows):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
```

```
        :return: (list) of dictionary rows
        """
        list_of_rows.clear()  # clear current data
        try:
            file = open(file_name, "rb")
            for i in range(3):
                line = pickle.load(file)
                task, priority = line.split(",")
                # Here we make sure to convert all text to lower case to match with later
                # functionality.
                row_dic = {"Task": task.strip().lower(), "Priority": priority.strip().lower()}
                list_of_rows.append(row_dic)
            file.close()
            return list_of_rows
        except:
            print("No file found. You may add tasks and a new file will be created.")

    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        """ Adds user-defined task/priority items into a list of dictionary rows

        :param task: (string) user-defined task name:
        :param priority: (string) user-defined priority for task:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        # insert into data table
        list_of_rows.append({"Task": task, "Priority": priority})
        return list_of_rows

    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        """ Removes user-defined task/priority items from a list of dictionary rows

        :param task: (string) user-defined task name:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        # first set removed parameter as false, then loop through table and identify item to be removed
        itemRemoved = False
        for row_dic in list_of_rows:
            if row_dic["Task"] == task.strip():
                list_of_rows.remove(row_dic)
                print("Item removed!")
                # Here we must set itemRemoved to True to not print the following block
                itemRemoved = True
        # if item not found, print error message to user
        if itemRemoved == False:
            print("Item not found.")

        return list_of_rows

    @staticmethod
    def write_data_to_file(file_name, list_of_rows):
        """ Write list of user-defined task/priority items to file

        :param file_name: (string) user-defined filename for output:
        :param list_of_rows: (list) you want filled with file data:
        :return: nothing
        """
        # Process the data into a file
        outfile = open(file_name, "w")
        for row_dic in list_of_rows:
            outfile.write(row_dic["Task"] + "," + row_dic["Priority"] + "\n")
        outfile.close()
        # Display a message to the user
        print("Data saved to file!")
        # Print extra line for space
        print()

    @staticmethod
    def pickle_that_file(file_name, list_of_rows):
        """ Write data to a pickle.

        :param file_name: (string) user-defined filename for pickle output:
        :param list_of_rows: (list) you want filled with file data:
        :return: nothing
        """
        # Process the data into a file
        outfile = open(file_name, "wb")
        for row_dic in list_of_rows:
            pickle.dump(row_dic["Task"] + "," + row_dic["Priority"] + "\n", outfile)
        outfile.close()
        # Display a message to the user
```

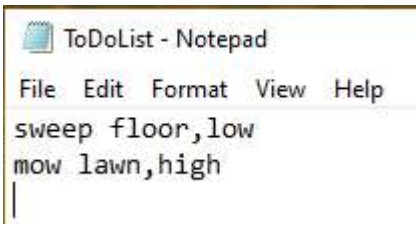
```
        print("File has been pickled!")
        # Print extra line for space
        print()

# Presentation (Input/Output)  ------------------------------------------- #


class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_tasks():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print('''
        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program
        ''')
        print()  # Add an extra line for looks

    @staticmethod
    def input_menu_choice():
        """ Gets the menu choice from a user

        :return: string
        """
        try:
            choice = str(input("Which option would you like to perform? [1 to 6] - ")).strip()
            print()  # Add an extra line for looks
            return choice
        except:
            print("Please enter a numeric value between 1 and 6.")
            print()  # Add an extra line for looks

    @staticmethod
    def output_current_tasks_in_list(list_of_rows):
        """ Shows the current Tasks in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current tasks ToDo are: *******")
        for row_dic in list_of_rows:
            print(row_dic["Task"] + " (" + row_dic["Priority"] + ")")
        print("*****************************************")
        print()  # Add an extra line for looks

    @staticmethod
    def input_new_task_and_priority():
        """ Prompts user for new input (task and priority) to add to list

        :return: (tuple) task and string as a 2-element tuple:
        """
        print("Please enter a to-do item and its priority: ")
        task = input("Enter a to-do item: ").strip().lower()
        priority = input("Enter a priority: ").strip().lower()
        # return items as tuple
        return (task, priority)

    @staticmethod
    def input_task_to_remove():
        """ Prompts user for new input (task and priority) to add to list

        :return: (string) task:
        """
        # prompt user for task to remove
        print("Please enter the item you would like to remove: ")
        task = input("Enter name here: ")
        # return task
        return task.strip().lower()


# Main Body of Script  ---------------------------------------------------- #

# Step 1 - Display a menu of choices to the user
while (True):
    # Step 2 Show current data
```

```
    IO.output_current_tasks_in_list(table_lst)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    # Step 3 - Process user's menu choice
    if choice_str.strip() == '1': # Load data
        Processor.read_data_from_file(table_lst)  # read file data

    elif choice_str.strip() == '2':  # Add a new Task
        # First get new input from user (a task and a priority)
        (task, priority) = IO.input_new_task_and_priority()
        # Now add that data to the list
        Processor.add_data_to_list(task, priority, table_lst)
        continue  # to show the menu

    elif choice_str == '3':  # Remove an existing Task
        # First ask user which item to remove
        task = IO.input_task_to_remove()
        # Now remove that item
        Processor.remove_data_from_list(task, table_lst)
        continue  # to show the menu

    elif choice_str == '4':  # Save Data to File
        # Here we write to file using pre-defined filename
        Processor.write_data_to_file(file_name_str, table_lst)
        continue  # to show the menu

    elif choice_str == '5': # Save Data to Pickle
        # We pickle the text file
        Processor.pickle_that_file(file_name_str_p, table_lst)
        continue  # to show the menu

    elif choice_str == '6':  # Exit Program
        print("Goodbye!")
        break  # and Exit
```

The script first tries to read in a list of to-do items based on a relative path. If no list is found, it prints an error message to the user, but then continues. It next prompts the user with a menu and waits for input. The user may choose to add items to the list, to remove an item, to write to an output (text) file, or to exit.

Part 2: Running the Script

The next part of this assignment was to run the script I had just created. I first ran it directly in the PyCharm IDE, starting with a list file with the following items as a text file:



After reading this file in from the text file, I added a new item called “wash dishes,” as shown below:

Screenshot 1 of 3

```
C:\_PythonClass\Assignment_07\venv\Scripts\python.exe C:/_PythonClass/Assignment_07/
******* The current tasks ToDo are: *******
*******************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 6] - 1

Please enter name of existing file to read in:ToDoList.txt
******* The current tasks ToDo are: *******
sweep floor (low)
mow lawn (high)
*******************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 6] - 2
```

I then wrote the resulting list back to a text file, as shown below in Screenshot 2:

Screenshot 2 of 3:

```
Which option would you like to perform? [1 to 6] - 2

Please enter a to-do item and its priority:
Enter a to-do item: wash dishes
Enter a priority: low
******* The current tasks ToDo are: *******
sweep floor (low)
mow lawn (high)
wash dishes (low)
*****************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 6] - 4

Data saved to file!
```

After running the script, the text file now looked like this:

ToDoList - Notepad

File Edit Format View Help

```
sweep floor,low
mow lawn,high
wash dishes,low
```

I then chose to also write the list to a pickle file by choosing Option 5, and then reading that pickle file back in as shown below:

Screenshot 3 of 3:

```
Which option would you like to perform? [1 to 6] - 5

File has been pickled!

******* The current tasks ToDo are: *******
sweep floor (low)
mow lawn (high)
wash dishes (low)
*****************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 6] - 1

Please enter name of existing file to read in:ToDoList.pickle
******* The current tasks ToDo are: *******
sweep floor (low)
mow lawn (high)
wash dishes (low)
*****************************************
```

Interestingly, when looking at the raw pickle file, it looks like this (below), which shows that pickling does indeed change the encoding by “flattening” the data into a stream of bytes:

```
The file was loaded in a wrong encoding: 'UTF-8'
1  ⏎EOT⏎DC4NULNULNULNULNULNULNUL⏎DLE
2  ⏎.⏎EOT⏎DC2NULNULNULNULNULNULNUL⏎
3  ⏎.⏎EOT⏎DC4NULNULNULNULNULNULNUL⏎
4  ⏎.
```

I then opened a command terminal window by typing “cmd” in the file directory bar. I then typed in “python Assignment_07.py” to execute the script located within this folder. For this part, I chose to read in the data from the pickle file I had just created. The results of the script are shown below:

Screenshot 1 of 2:

```
C:\Windows\System32\cmd.exe - python  Assignment_07.py
Microsoft Windows [Version 10.0.19042.1348]
(c) Microsoft Corporation. All rights reserved.

C:\_PythonClass\Assignment_07>python Assignment_07.py
******* The current tasks ToDo are: *******
*******************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 6] - 1

Please enter name of existing file to read in:ToDoList.pickle
******* The current tasks ToDo are: *******
sweep floor (low)
mow lawn (high)
wash dishes (low)
*******************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program
```

I also checked that the structured error handling enhancements were working correctly by trying to read in a file that did not exist. As shown in Screenshot 2 below, this threw an error message back to the user:

Screenshot 2 of 2:

```
Which option would you like to perform? [1 to 6] - 1

Please enter name of existing file to read in:DoesNotExist.txt
File not found!
******* The current tasks ToDo are: *******
********************************************


        Menu of Options
        1) Read in Data
        2) Add a new Task
        3) Remove an existing Task
        4) Save Data to File
        5) Save Data to Pickle File
        6) Exit Program
```

Posting to Github

Files were posted to the new repository located here: https://github.com/mah765/IntroToProg_Assignment07

Conclusions and Observations

In this assignment we continued our work with functions, dictionaries, and input/output, but in this exercise we enhanced the usability and flow of the script by adding additional structured error handling and incorporating the ability to read and write to pickle files. Structured error handling is an important concept in programming, and makes a program much more usable by providing meaningful error messages to the end user (as opposed to system generated messages that are meant for the developer). Pickling is an interesting concept that is not unique to Python. It is essentially the process of converting a Python object to a byte stream, which can then be stored or transmitted and then "re-incarnated" in another script. It is important to note that although pickling appears to be encoding information in an illegible manner, this is actually not the case, as it can be easily reconstructed by another Python script.