Alex Jehle

August 24th, 2020

IT FDN 110 A

Assignment 07

GitHub: <https://github.com/anjehle/ITFnd100-Mod07>

To Do List Script with Pickles and Error Handling

# Introduction

The purpose of this document is to familiarize the reader with the concepts of pickling and exception handling in Python. This is done using the program from the Mod06 Assignment with variations made. The original program can be accessed here: <https://github.com/anjehle/IntroToProg-Python-Mod06>

# Pickling

Pickling is a way for python users to condense data using serialization during the reading and loading process. By unloading data through pickles the user is converting their file into a binary protocol using a byte stream. Inversely, when they unload data, or “unpickle”, they are converting that binary file into the traditional object hierarchy. This process is as simple as altering the way the read and write commands are written. Instead of simply using the read command, a special pickle.load([File to load]) command is used, as seen in Figure 1. Additionally, to write to a pickled file the pickle.dump([data to be written], [File]) command is used as seen in Figure 2. Apart from these changes, no other edits are necessary, making this a convenient and easily executable change to help with any basic file!

## Application with To-Do List Program

Pickles were incorporated into the previous program from the Mod06 assignment. This was used to pickle and unpickle the to-do list via the code shown in Figure 1 and Figure 2.

def read\_data\_from\_file(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:  
 objFile = open(file\_name, 'rb')  
 print('File found!')  
 list\_of\_rows = pickle.load(objFile)  
 objFile.close()  
 except:  
 print('No existing to-do list available')  
 return list\_of\_rows, 'Success'

Figure 1: Reading data from the to-do list using pickles

def write\_data\_to\_file(file\_name, list\_of\_rows):  
 *""" Writes data from list of dictionary rows to file  
 :param file\_name: (string) with name of file:  
 :param list\_of\_rows: (list) you want added to text file:  
 :return: (list) of dictionary rows  
 """* objFile = open(file\_name, 'wb')  
 pickle.dump(list\_of\_rows, objFile)  
 objFile.close()  
 return list\_of\_rows, 'Success'

Figure 2: Unloading data from the to-do list using pickles

## Execution in Pycharm

The program with the code modifications shown above was executed in PyCharm. The program was successfully pickled and unpickled, as shown in Figure 3.

# 

Figure 3: Execution of pickled file in PyCharm

## Execution in Terminal

Immediately after execution in PyCharm, the program was ran via the terminal. As you can see in Figure 4, the changes made in the PyCharm execution were successfully saved and unpickled.

# 

Figure 4: Execution of pickled file in the terminal

# Exception and Error Handling

Effectively using exceptions and error handling in python can be a powerful way to make your code more robust. Although this document will only cover a basic example of this, they are very modifiable. The programmer is able to alter these exceptions to encapsulate a vast range of errors that would normally stall out a program. These can be user defined (using a similar method as defining a class) or more traditional try/except cases. In this example, a try: was added to the program as seen in Figure 5. This alteration checks to see if a file exists before it is opened. If it does not exist the except: line is ran. The user is notified whichever case is ran by outputting either “File found!” or “No existing to-do list available”.

def read\_data\_from\_file(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:  
 objFile = open(file\_name, 'rb')  
 print('File found!')  
 list\_of\_rows = pickle.load(objFile)  
 objFile.close()  
 except:  
 print('No existing to-do list available')  
 return list\_of\_rows, 'Success'

Figure 5: A file load with a try: command in it

## Execution in Pycharm

The program with the code modifications shown above was executed in PyCharm. The program successfully identified that no file was found. When a task was saved, it successfully recognized the new file.

# 

Figure 6: Execution of try/except program in PyCharm

## Execution in Terminal

The same example as above was ran in the terminal next after deleting the .dat file. The program successfully identified that no file was found as seen in Figure 7. When a task was saved, it successfully recognized the new file. All of these resources were selected because they have a basic explanation but wide range of what they cover. The examples are clear and were found to be completely accurate and executable. In general, they provide a very insightful introduction but shouldn’t be referenced for advanced instruction on the concepts.

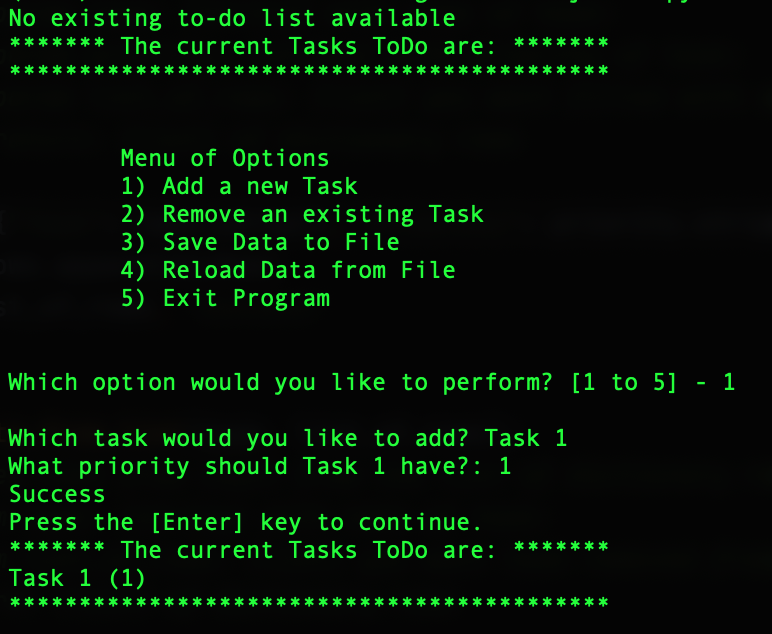
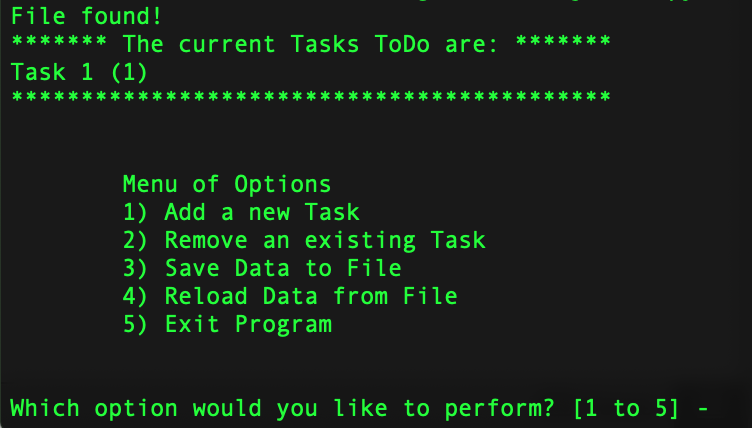
 

Figure 7: Execution of try/except program in the terminal

# Summary

Although these were both very basic examples, there is much more to learn about pickles and error handling in python. The webistes cited in the ‘Resources’ section below expand on these concept and provide much more information and examples of various use cases.

# Appendix I: Full Python Script

# ---------------------------------------------------------------------------- #  
# Title: Assignment 07  
# Description: Working with functions in a class,  
# When the program starts, load each "row" of data  
# in "ToDoToDoList.txt" into a python Dictionary.  
# Add the each dictionary "row" to a python list "table"  
# ChangeLog (Who,When,What):  
# RRoot,1.1.2030,Created started script  
# RRoot,1.1.2030,Added code to complete assignment 5  
# AJehle,8.11.2020,Modified code to complete assignment 6  
# AJehle,8.23.2020,Modified code to complete assignment 7  
# ---------------------------------------------------------------------------- #  
  
# Data ---------------------------------------------------------------------- #  
import pickle  
# Declare variables and constants  
strFileName = "ToDoFile.dat" # The name of the data file  
objFile = None # An object that represents a file  
dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}  
lstTable = [] # A list that acts as a 'table' of rows  
strChoice = "" # Captures the user option selection  
strTask = "" # Captures the user task data  
strPriority = "" # Captures the user priority data  
strStatus = "" # Captures the status of an processing functions  
  
  
# Processing --------------------------------------------------------------- #  
class Processor:  
 *""" Performs Processing tasks """* @staticmethod  
 def read\_data\_from\_file(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:  
 objFile = open(file\_name, 'rb')  
 print('File found!')  
 list\_of\_rows = pickle.load(objFile)  
 objFile.close()  
 except:  
 print('No existing to-do list available')  
 return list\_of\_rows, 'Success'  
  
 @staticmethod  
 def add\_data\_to\_list(task, priority, list\_of\_rows):  
 *""" Adds task from input to the list of dictionary rows  
 :param task: (string) with name of task:  
 :param priority: (string) with priority of task:  
 :param list\_of\_rows: (list) you want filled with additional task:  
 :return: (list) of dictionary rows  
 """* dicRow = {"Task": task.strip(), "Priority": priority.strip()}  
 list\_of\_rows.append(dicRow)  
 return list\_of\_rows, 'Success'  
  
 @staticmethod  
 def remove\_data\_from\_list(task, list\_of\_rows):  
 *""" Removes task from input from the list of dictionary rows  
 :param task: (string) with name of task:  
 :param list\_of\_rows: (list) you want task removed from:  
 :return: (list) of dictionary rows  
 """* boolfound = False  
 for dicRow in list\_of\_rows:  
 if dicRow["Task"] == task:  
 list\_of\_rows.remove(dicRow)  
 boolfound = True  
 if boolfound == False:  
 print('Task not found')  
 return list\_of\_rows, 'Success'  
  
 @staticmethod  
 def write\_data\_to\_file(file\_name, list\_of\_rows):  
 *""" Writes data from list of dictionary rows to file  
 :param file\_name: (string) with name of file:  
 :param list\_of\_rows: (list) you want added to text file:  
 :return: (list) of dictionary rows  
 """* objFile = open(file\_name, 'wb')  
 pickle.dump(list\_of\_rows, objFile)  
 objFile.close()  
 return list\_of\_rows, 'Success'  
  
  
# Presentation (Input/Output) -------------------------------------------- #  
class IO:  
 *""" Performs Input and Output tasks """* @staticmethod  
 def print\_menu\_Tasks():  
 *""" Display a menu of choices to the user  
 :return: nothing  
 """* print('''  
 Menu of Options  
 1) Add a new Task  
 2) Remove an existing Task  
 3) Save Data to File   
 4) Reload Data from File  
 5) Exit Program  
 ''')  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def input\_menu\_choice():  
 *""" Gets the menu choice from a user  
 :return: string  
 """* choice = str(input("Which option would you like to perform? [1 to 5] - ")).strip()  
 print() # Add an extra line for looks  
 return choice  
  
 @staticmethod  
 def print\_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 dicRow in list\_of\_rows:  
 print(dicRow["Task"] + " (" + dicRow["Priority"] + ")")  
 print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def input\_yes\_no\_choice(message):  
 *""" Gets a yes or no choice from the user  
 :return: string  
 """* return str(input(message)).strip().lower()  
  
 @staticmethod  
 def input\_press\_to\_continue(optional\_message=''):  
 *""" Pause program and show a message before continuing  
  
 :param optional\_message: An optional message you want to display  
 :return: nothing  
 """* print(optional\_message)  
 input('Press the [Enter] key to continue.')  
  
 @staticmethod  
 def input\_new\_task\_and\_priority():  
 pass # *TODO: Add Code Here!* task = str(input("Which task would you like to add? ")).strip()  
 priority = str(input('What priority should ' + task + ' have?: ')).strip()  
 return task, priority  
  
 @staticmethod  
 def input\_task\_to\_remove():  
 pass # *TODO: Add Code Here!* task = str(input("Which task would you like to remove? ")).strip()  
 return task  
  
  
# Main Body of Script ------------------------------------------------------ #  
  
# Step 1 - When the program starts, Load data from ToDoFile.txt.  
lstTable, success = Processor.read\_data\_from\_file(strFileName, lstTable) # read file data  
  
# Step 2 - Display a menu of choices to the user  
while (True):  
 # Step 3 Show current data  
 IO.print\_current\_Tasks\_in\_list(lstTable) # Show current data in the list/table  
 IO.print\_menu\_Tasks() # Shows menu  
 strChoice = IO.input\_menu\_choice() # Get menu option  
  
 # Step 4 - Process user's menu choice  
 if strChoice.strip() == '1': # Add a new Task  
 task, priority = IO.input\_new\_task\_and\_priority()  
 lstTable, strStatus = Processor.add\_data\_to\_list(task, priority, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu  
  
 elif strChoice == '2': # Remove an existing Task  
 task = IO.input\_task\_to\_remove()  
 lstTable, strStatus = Processor.remove\_data\_from\_list(task, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu  
  
 elif strChoice == '3': # Save Data to File  
 strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")  
 if strChoice.lower() == "y":  
 lstTable, strStatus = Processor.write\_data\_to\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("Save Cancelled!")  
 continue # to show the menu  
  
 elif strChoice == '4': # Reload Data from File  
 print("Warning: Unsaved Data Will Be Lost!")  
 strChoice = IO.input\_yes\_no\_choice("Are you sure you want to reload data from file? (y/n) - ")  
 if strChoice.lower() == 'y':  
 lstTable, strStatus = Processor.read\_data\_from\_file(strFileName, lstTable)  
 IO.input\_press\_to\_continue(strStatus)  
 else:  
 IO.input\_press\_to\_continue("File Reload Cancelled!")  
 continue # to show the menu  
  
 elif strChoice == '5': # Exit Program  
 print("Goodbye!")  
 break # and Exit  
 else:  
 print("Valid option not selected.")  
 continue

# Resources

P. (2017, July 20). How to Pickle: A Beginner's Guide to Pickling and Unpickling. Retrieved August 24, 2020, from https://www.pythoncentral.io/how-to-pickle-unpickle-tutorial/

P. (2020). 8. Errors and Exceptions. Retrieved August 24, 2020, from <https://docs.python.org/3/tutorial/errors.html>

P. (n.d.). Python Exception Handling Using try, except and finally statement. Retrieved August 24, 2020, from https://www.programiz.com/python-programming/exception-handling

P. (n.d.). Python Pickle Module for saving Objects by serialization. Retrieved August 24, 2020, from <https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>

Z0o0p, G., Z0o0p, & Geeks for Geeks. (2020, July 26). Read List of Dictionaries from File in Python. Retrieved August 24, 2020, from https://www.geeksforgeeks.org/read-list-of-dictionaries-from-file-in-python/