Geoffrey Sears  
November 20, 2019  
Foundations of Programming, Python  
Assignment 07

<https://github.com/geoffsears/IntroToProg-Python-Mod07>

# Introduction

The assignment this week was to create a script which demonstrated pickling data, and use try/except statements to handle predictable exceptions to your code.

This document walks through the steps taken to complete this assignment. I have included a summary at the end of the document to talk about what I learned and how I applied the lessons of this week.

# Apply Your Knowledge

I initially tried to update last week’s assignment to use Pickle, but I decided that was going to introduce too much work in trying to adjust the data handling which was implicit to using text files to be replaced by more direct object handling used by Pickle. As a result, I went back to older assignments and decided the best way to demonstrate these concepts was through a simpler script focused on collecting terms into a list and writing that list to a binary file for later usage.

The steps I followed in defining this script were:

1. I wrote a block statement at the top of my script describing the objectives for the program. In this case I stated that I wanted a program which collected elements of a list, read/wrote that list to a binary file using Pickle, and provided error handling through the use of Try…Except statements.
2. I then Separated Concerns between the data, processing and presentation sections to shape future development.
3. On deciding that I should have list management functionality (add/remove/show) I decided I should make a Menu and write functions for each list management activity.
   1. I elected to not use classes because of the relative simplicity of the script. I could have followed the example of last week and broken the functions between listProcessing and IO, but decided against this approach
4. I then wrote out the main while loop of the program and tested each element in the menu, later adding an option for exiting the program and removing items added to the list.
5. Finally, I tried to trigger different failure modes and wrote in code to handle them as Try…Except statements.

The first statement in my script:

**import** pickle

Imported the pickle module. This permits the writing of string, numeric, tuple, list, and dictionary objects to a binary file. In my script I am only “pickling” one object (the list lstWords), but pickle in combination with “shelve” allows for the option of bundling multiple options into the same binary file for later loading.

## Data Section

The Data section of my script:

*# Data*lstWords = []  
wordList = []  
choice = **''**dataFile = **'WordList.dat'**

Initialized the variables that later would be called by the script. The script requires that WordList.dat be present for it to work. More on this and error handling issues I ran into concerning this later in this document under “Main Script”

## Processing Section

The processing section of my script has a series of functions described below:

**def** collectWords():  
 *"""Collects terms to be added to the word list.* **:return***: string newWord"""* newWord = input(**'Enter the next term in your list:'**)  
 **return** newWord

The collectWords() function grabs user input in the form of a string and returns it as a new word to be appended to the lstWords list via the block of statements under “Choice 1” in the main while loop of the script.

**def** mainMenu():  
 *"""Displays the main menu for the application, and choices which can be selected  
 to execute different blocks of code.* **:return***: nothing"""* print(**'''Select an Option from the Items below:  
 1) Add word to the list  
 2) Write list to file  
 3) Read words from list  
 4) Remove words from list  
 5) Show current list  
 6) Exit Program'''**)

The mainMenu() function displays the menu of options to the user, and is invoked from within the main While loop of the script, to remind the user of the numeric choices they can make to execute different aspects of the program.

**def** writeFile(wordList,dataFile):  
 *"""Writes data collected into the word list to the designated file.* **:param***: wordlist - a list of terms to be added to file* **:param***: dataFile - the name of the file which is being written to.* **:return***: none"""* f = open(dataFile,**"wb"**)  
 pickle.dump(wordList,f)  
 f.close()  
 print()  
 print(**'Your data has been written to a binary file named "WordList.dat"'**)  
 print()

The writeFile() function is the first point at which we make use of “Pickle” in our code. This function takes the current word list, and provided file name and opens a write binary connection to the file. This means that the script will overwrite the contents of the WordList.dat file with binary code which allows Python to address this file via subsequent statements as a list, or to load the list without other processing.

The pickle.dump(wordlist, f) statement is where we are writing the provided list to the data file whose open connection has been assigned to the variable “f” for the purpose of this statement. This statement will execute whether there is a file named “WordList.dat” or not. In the absence of that file it will overwrite the contents and replace with the ones currently stored in memory.

**def** readFile(dataFile):  
 *"""Reads data from stored binary file 'WordList.dat'* **:param***: datafile (filename) used to designate were to read from.* **:return***: lstWords (list) of terms read from dataFile"""* **try**:  
 f = open(dataFile, **"rb"**)  
 lstWords = pickle.load(f)  
 f.close()  
 print()  
 print(lstWords)  
 print()  
 **return** lstWords  
 **except**:  
 f = open(dataFile, **"ab"**)  
 lstWords = []  
 f.close()  
 **return** lstWords

The readFile() function reads the contents of the “WordList.dat” file into memory as a list. Because Pickle allows us to save objects as they are typed within Python, when we use the “pickle.load()” command we are provided that object in this case a list as it is typed.

The open command addresses the provided “dataFile” in the “rb” mode which means it is opening a read-only connection to a binary file. If the file does not exist, the program will throw an exception at this point. I introduced a try..except statement block force the program to create an empty data file in the case that one did not exist through the use of the IOError generated by the “rb” method when the pickle.load command could not read the missing file.

**def** removeItem(term,lstWords):  
 *"""Removes items from the active list.* **:param***: term (string) to be added to lstWords* **:param***: lstWords(list) container for terms provided to the list.* **:return***: lstWords (list) """* **try**:  
 lstWords.remove(term)  
 **except** ValueError:  
 print()  
 print(**'The term,'**, term, **'is not in the list. Check your spelling and punctuation.'**)  
 **return** lstWords

The “removeItem()” function was designed to remove elements from the Word List maintained by the program. It is the first place I thought to put a Try … Except statement. The function accepts a provided term, and the currently stored list in memory. It tries to remove that term from the lstWords list, and if a ValueError is raised, it handles that by telling the user that the item requested to be removed is not in the list. This will then bring them back to the main menu, instead of dropping them to a prompt.

## Main Loop

The main loop of the script loads the existing data stored in WordList.dat through the readFile() function and then proceeds through a While loop which iteratively displays the main menu, collects an input from the user, and then executes the associated action using the defined function associated to that choice.

The first statement of this section:

lstWords = readFile(dataFile)

Initializes the script with the stored binary file. It invokes the readFile() function which makes uses of Pickle to load the contents of the file as a list which is then assigned to lstWords for use by the script. This is much more compact than having to sequentially walk through the contents of the saved text file which we did in the previous assignment.

The next section contains the main while loop which walks through the different selections the user can make off of the main menu:

**while True**:  
 mainMenu()  
 choice = int(input(**'Please select a number between 1 and 6:'**))  
 **try**:  
 **if** choice == 1:  
 lstWords.append(collectWords())  
 print(lstWords)  
 **elif** choice == 2:  
 writeFile(lstWords, dataFile)  
 **elif** choice == 3:  
 lstWords = readFile(dataFile)  
 **elif** choice == 4:  
 print(lstWords)  
 term = input(**'What term do you want to remove?'**)  
 lstWords = removeItem(term, lstWords)  
 print()  
 print(lstWords)  
 **elif** choice == 5:  
 print(**'Your current list is:'**)  
 print()  
 print(lstWords)  
 **elif** choice == 6:  
 print(**'Thank you.'**)  
 **break  
 else**:  
 print()  
 **raise  
 except**:  
 print(**'You need to make a section between 1 and 6'**)  
 print()

Each successive choice invokes a separate function which has already been described. Some elements which might have been moved into a function, such as choice 4 had their input command invoked from the menu before calling the function. Despite Separation of Concerns as a principle, I chose to do this because of the economy of code needed to collect this simple content.

Initially I only had three menu items to cover the minimum requirements of the assignment, but added choices to remove members of the list because it presented another opportunity to make use of try… except statements, an option to view the list as well as providing an option to exit the script.

Finally I decided to introduce a try..except statement block within this while loop to capture situations where the user decided to enter a value outside of 1-6 as a response to the menu. The raise statement was used to trigger the except block. This permits the user to generate an exception without the script closing out.

## Demonstrating the Script

The following copied text demonstrates the script working with PyCharm and the command line.

### PyCharm

On first execution the script presents a menu asking the user to select a number from 1 to 6:

C:\Python3-7-4\\_PythonClass\Assignment07\venv\Scripts\python.exe C:/Python3-7-4/\_PythonClass/Assignment07/Assignment07a.py

Select an Option from the Items below:

1) Add word to the list

2) Write list to file

3) Read words from list

4) Remove words from list

5) Show current list

6) Exit Program

Please select a number between 1 and 6:

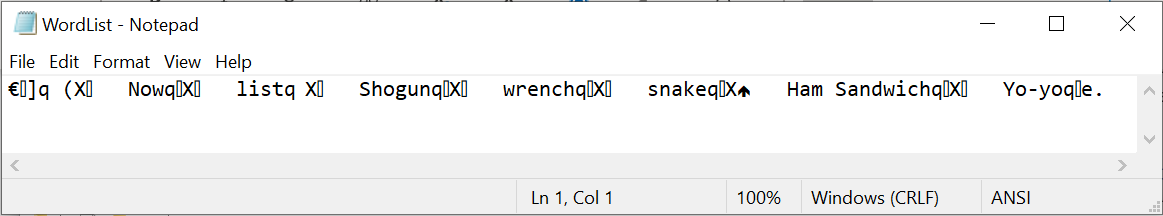
If you select option #1 the script will ask you to enter the next term in the list, and then add that value to the list which is displayed below as a printed list:

Please select a number between 1 and 6:1

Enter the next term in your list:Yo-yo

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo']

If you select open #2 the script will write the current list to a binary file. The contents of which are confirmed below in a screenshot of the file. Note that the contents reflect binary content, as opposed to raw text .



Option #3:

Please select a number between 1 and 6:3

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo']

Option #3 reads the saved binary file a list of elements to be included in the lstWords list stored in memory. It does this using the pickle.load() command.

Option #4

Please select a number between 1 and 6:4

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo']

What term do you want to remove?wrench

['Now', 'list', 'Shogun', 'snake', 'Ham Sandwich', 'Yo-yo']

When option #4 is chosen, the program displays the elements in the list, and then asks you to provide the value which you want to remove. This step has a try…except block which is there to trap cases where you provide a value which is not in the provided list:

Please select a number between 1 and 6:4

['Now', 'list', 'Shogun', 'snake', 'Ham Sandwich', 'Yo-yo']

What term do you want to remove?LIST

The term, LIST is not in the list. Check your spelling and punctuation.

['Now', 'list', 'Shogun', 'snake', 'Ham Sandwich', 'Yo-yo']

Option #5 and #6

Both of these options have basic functionality to print the list to console and to exit the program.

### Windows prompt

The following capture performs the same tasks as displayed above for Pycharm.

c:\Python3-7-4\\_PythonClass\Assignment07>python.exe Assignment07a.py

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo']

<Menu>

Please select a number between 1 and 6:1

Enter the next term in your list:stack

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo', 'stack']

<Menu>

Please select a number between 1 and 6:2

Your data has been written to a binary file named "WordList.dat"

<Menu>

Please select a number between 1 and 6:3

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo', 'stack']

<Menu>

Please select a number between 1 and 6:4

['Now', 'list', 'Shogun', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo', 'stack']

What term do you want to remove?Shogun

['Now', 'list', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo', 'stack']

<Menu>

Please select a number between 1 and 6:5

Your current list is:

['Now', 'list', 'wrench', 'snake', 'Ham Sandwich', 'Yo-yo', 'stack']

<Menu>

Please select a number between 1 and 6:6

Thank you.

c:\Python3-7-4\\_PythonClass\Assignment07>

# Summary

The assignment this week involved writing a script which demonstrated the functionality introduced via the pickle module and use of standard error handling. I created a trimmed down version of the application we developed over the last couple of assignments to demonstrate the use of the Pickle module for the purpose of reading and writing data from a binary file which served as a container for the list object which was central to this application. This has the advantage of requiring less code to dump the list to the data file, and to load its contents as a list back into memory within the program.

Error handling took a couple different forms in this program. The first was trapping potential value errors in the “Remove Item from List” option. It trapped the possibility of the user entering values which were not on the list stored in memory, which would normally result in an exception which would disrupt the operation of the program. Then I introduced another try…except block in the main while loop of the program to ensure that the user was entering a number between 1 and 6 to select an option for the script to execute. This worked by executing a “raise” statement when the script walked through each of the options numbered 1 through 6. When the raise statement is hit, the script jumps to the except: block which resulted in a message to the user about limiting their responses to be between 1 and 6 as well as allowing the program to proceed back to the top of the while loop. Finally I included a try…except statement block to trap the possibility of the program failing if the “WordList.dat” file being missing when the program was initialized. This block took advantage of the throwing of an error by the “rb” mode of opening the file to cause it instead to use “wb” mode. When you attempt to write to a non-existent file using “wb” mode, it will create the file. In order to permit the program to load I then added the assignment of an empty list to lstWords, which permits the program to proceed through its normal operation.

Things are beginning to come together over the last few assignments, in that I was able to pull this code together fairly quickly to demonstrate the Pickle module and error control. It was much more efficient to code storing the list object into a binary file than it was in the previous operations by removing the need to parse a text file and use loops to extract information into the desired format.

Trying to think of errors to trap was a new exercise this week. Some presented themselves readily, such as the IOError for having a missing WordList.dat file, but others had to be conceptualized on behalf of the hypothetical user which became something of a game. The simple structure of this program limited the number of options for error handling, but I can see how this would be very important for developing more complex applications.