Craig Muth

11/26/2022

IT FDN 110 A Au 22: Foundations Of Programming: Python

Assignment 07

https://github.com/cjmuth/IntroToProg-Python-Mod07

Exception Handling and the pickle module

Introduction

In this project we will create a program that will read from a pickle file if present, and if not, will use an exception handler to prevent it from crashing and provide the user with an option to pull from alternate source. After the the data has been loaded, the user will be given the option to output to a pickle file.

Designing the program

For this example we will be using hard coded file names and structuring the code as functions. The data structure will be the same that was used for previous projects to create a todo list (a list of dictionaries) so we can focus on the new statements.

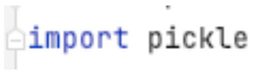
Logic for this program can be shown as:

- Try to load pickle file
- If file not found
    - Catch the exception and notify user
    - Provide option to load from text file
    - if yes
        - Open the text file
        - Read the data
        - Close the text file
        - Display the data
    - Else
        - Exit
- Else
    - Open file in binary read mode
    - Load the data
    - Close the file
    - Display the data
- Ask the user if they want to pickle the data in memory
- If yes
    - Open file in binary write mode
    - Dump the data to the file
    - Close the file
    - Exit
- Else

- Notify user data not pickled
- Exit

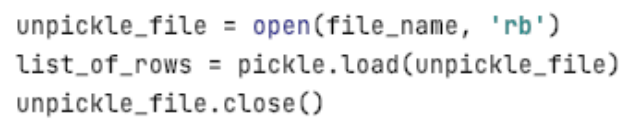
The ability to pickle a file is unique to Python. It's a method to convert data to a binary format and save this in a file with embedded formatting information (Elghamrawy, Karim (n.d.). *What is Pickling in Python? (In-depth Guide),* https://www.afternerd.com/blog/python-pickle/) (External site).

Before we can use any of its functions, we need to import the pickle module - which is simply



```
import pickle
```

Figure 1: Importing the pickle module

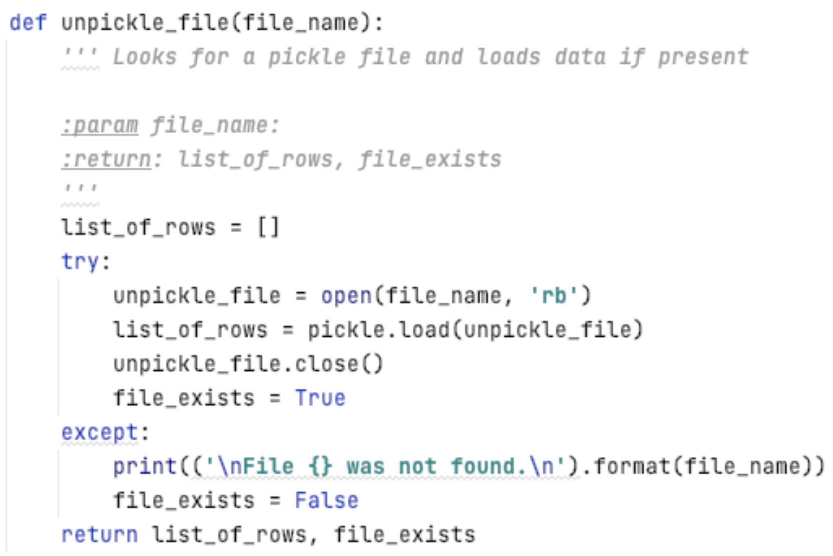


```
unpickle_file = open(file_name, 'rb')
list_of_rows = pickle.load(unpickle_file)
unpickle_file.close()
```

Figure 2: Unpickling the data

This should work as long as the file exists, but otherwise will cause the program to throw an exception (or runtime error) and crash. An exception occurs when the syntax of the code is correct, but the program encounters improper input (Kumar, Bijay (February 12, 2021). *Python Exceptions Handling (With Examples)*, **https://pythonguides.com/python-exceptions-handling/**) (External site). Users providing values of the wrong type, numbers that result in mathematical errors, or (as in this case) the program tries to read from a non existent source.

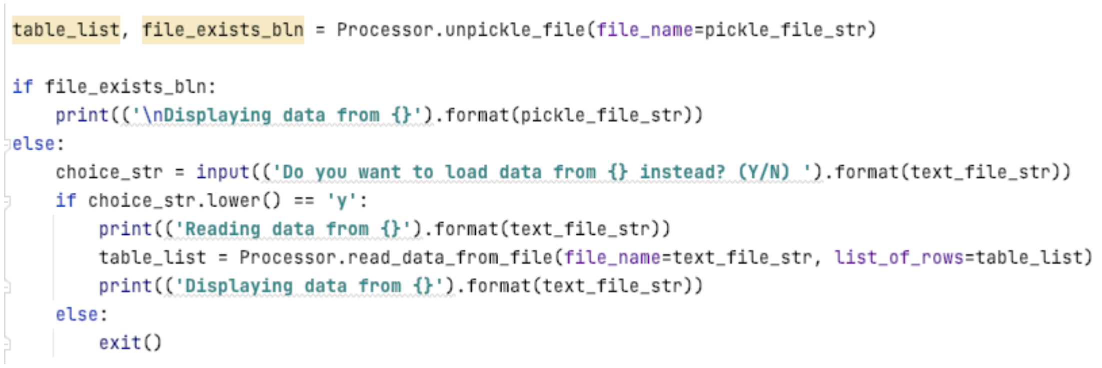
However, if we use a `try…except` statement, we can catch the exception and keep the program running. If we choose to, we can also return a value to a variable we can use to control the flow of the program, to notify the user and give them the option to pull data from a different source.



```
def unpickle_file(file_name):
    ''' Looks for a pickle file and loads data if present

    :param file_name:
    :return: list_of_rows, file_exists
    '''
    list_of_rows = []
    try:
        unpickle_file = open(file_name, 'rb')
        list_of_rows = pickle.load(unpickle_file)
        unpickle_file.close()
        file_exists = True
    except:
        print(('\nFile {} was not found.\n').format(file_name))
        file_exists = False
    return list_of_rows, file_exists
```

Figure 3: Unpickle function

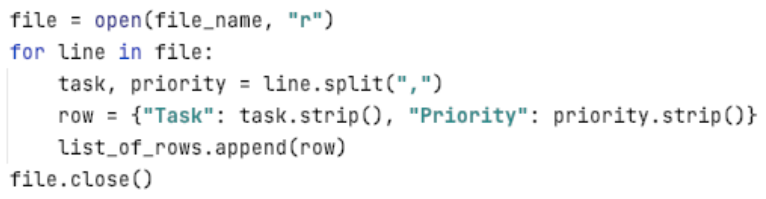
The exception statement could have been set up to capture and display the error as reported by the program by writing it as `except Exception as ex:` then printing the value of `ex` - I just chose to use my own statement instead. For reference, if the system statement were reported for this instance it would be `[Errno 2] No such file or directory: 'ToDoFile.pickle'.` In this case the usability of the system message is not much different than the custom message - but that may not be true in all cases, but the programmer has the option to use either depending on which they believe is more useful for their target users.



```
table_list, file_exists_bln = Processor.unpickle_file(file_name=pickle_file_str)

if file_exists_bln:
    print(('\nDisplaying data from {}').format(pickle_file_str))
else:
    choice_str = input(('Do you want to load data from {} instead? (Y/N) ').format(text_file_str))
    if choice_str.lower() == 'y':
        print(('Reading data from {}').format(text_file_str))
        table_list = Processor.read_data_from_file(file_name=text_file_str, list_of_rows=table_list)
        print(('Displaying data from {}').format(text_file_str))
    else:
        exit()
```

Figure 4: Calling the unpickle function, and option to call alternate data source

If we take a moment to look at the code for reading the text file, we can see it requires more lines of code than the pickle.load - which will help make the code cleaner and easier to maintain. We can also see that in order to create this function we had to know something about how the data was structured in the file in order to structure it correctly in memory.

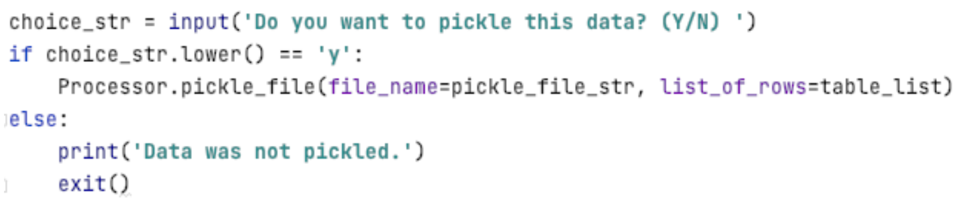


```
file = open(file_name, "r")
for line in file:
    task, priority = line.split(",")
    row = {"Task": task.strip(), "Priority": priority.strip()}
    list_of_rows.append(row)
file.close()
```

Figure 5: Reading the text file call alternate data source

Alternatively we could have read it as complete lines, review the data to determine the structure, then develop code to convert that into what we need for processing. Over time we could develop a list of functions to do the analysis for us, and launch one of several options to convert the data from one format to another. However, any time we encounter a file with a new structure it would mean having to write new code. With the pickle module, the file contains all the information the computer needs to extract the data into the format the originator intended.

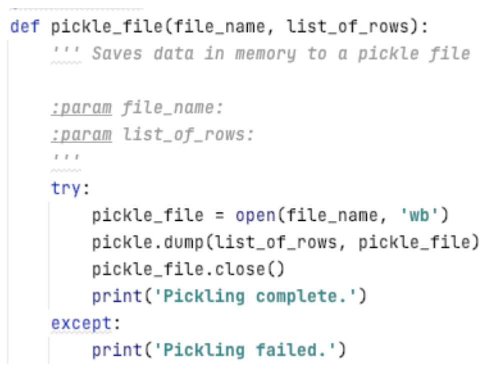
Now that the data is in memory, the only operation we're offering in the example is to save the data back in a pickled format.



```
choice_str = input('Do you want to pickle this data? (Y/N) ')
if choice_str.lower() == 'y':
    Processor.pickle_file(file_name=pickle_file_str, list_of_rows=table_list)
else:
    print('Data was not pickled.')
    exit()
```

Figure 6: Option to pickle or exit without pickling data

We can see that the method to write the data to a file is as simple as the method to load. We provide the data and the file name, and in the background the method will break down the formatting into machine readable data that's embedded in the output file.



```
def pickle_file(file_name, list_of_rows):
    ''' Saves data in memory to a pickle file

    :param file_name:
    :param list_of_rows:
    '''
    try:
        pickle_file = open(file_name, 'wb')
        pickle.dump(list_of_rows, pickle_file)
        pickle_file.close()
        print('Pickling complete.')
    except:
        print('Pickling failed.')
```

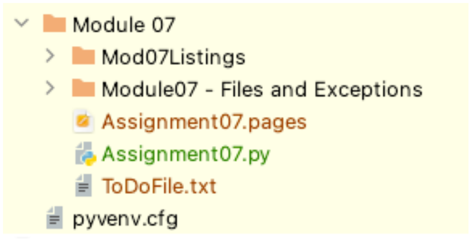
Figure 7: Pickle function

The file can then be passed along to other operators or systems, and as long as they have Python available they can extract it in the original structure without any additional information.

Running the program

Executing the program in Pycharm:

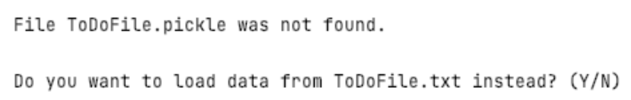
In order to test the error handling in the unpickle function we delete the pickle file to make sure there's no file for it to find.



```
Module 07
    Mod07Listings
    Module07 - Files and Exceptions
    Assignment07.pages
    Assignment07.py
    ToDoFile.txt
pyvenv.cfg
```

Figure 8: Ensuring no pickle file exists

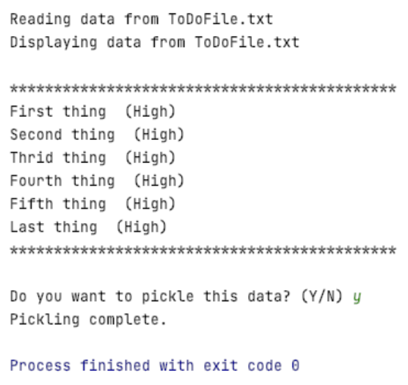
When we run the program, it states the file was not found and gives the option to load from the text file instead.



```
File ToDoFile.pickle was not found.

Do you want to load data from ToDoFile.txt instead? (Y/N)
```

Figure 9: Running without the pickle file

If we enter Y, the data is read and displayed. Then selecting to pickle the data we see the following.



```
Reading data from ToDoFile.txt
Displaying data from ToDoFile.txt

*********************************************
First thing  (High)
Second thing  (High)
Thrid thing  (High)
Fourth thing  (High)
Fifth thing  (High)
Last thing  (High)
*********************************************

Do you want to pickle this data? (Y/N) y
Pickling complete.

Process finished with exit code 0
```

Figure 10: Alternate source, and output a pickle file

Looking at the Pycharm Project window we can see the file has been created.

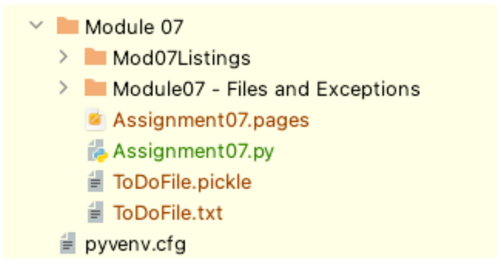
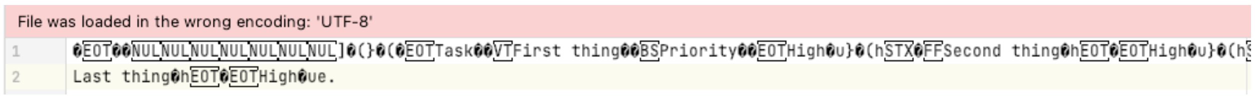


Figure 11: Pickle file created

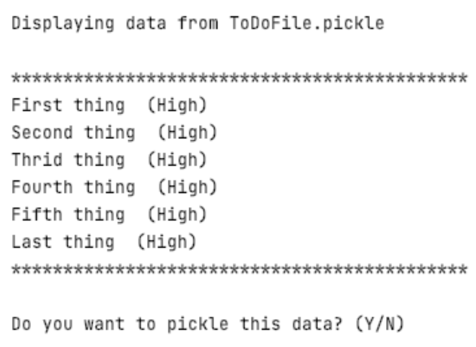
Opening the file we can see the data is almost human readable, but it is difficult. Changing the encoding might eliminate the error being reported here - but it would still be impractical (but not impossible) for human use.



```
File was loaded in the wrong encoding: 'UTF-8'
�EOT��NULNULNULNULNULNULNUL]�(}�(�EOTTask��VTFirst thing��BSPriority��EOTHigh�u}�(hSTX�FFSecond thing�hEOT�EOTHigh�u}�(h
Last thing�hEOT�EOTHigh�ue.
```

Figure 12: Viewing the contents of ToDoFile.pickle with Pycharm

If we run the program again with the pickle file present, we see the program successfully loaded the data and presented it back in much easier to read format.



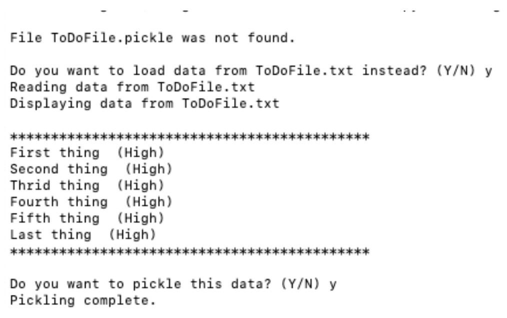
```
Displaying data from ToDoFile.pickle

*******************************************
First thing  (High)
Second thing  (High)
Thrid thing  (High)
Fourth thing  (High)
Fifth thing  (High)
Last thing  (High)
*******************************************

Do you want to pickle this data? (Y/N)
```

Figure 13: Running with the pickle file present

Whether the data was read from the text file or the pickle file, the program called the same function to display. Therefore indicating that even though we pulled from a different file format, the structure built into memory from the pickle file was the same as that which was loaded from the text file. So while the data in ToDoFile.pickle didn't look the same, in contained the information necessary for the computer to rebuild the original structure.

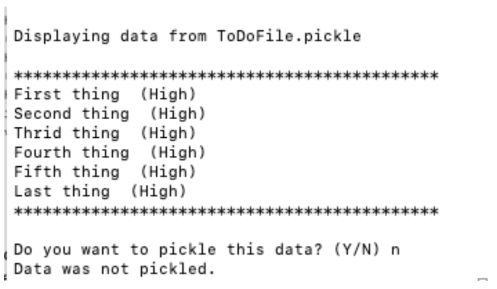
Executing in a Terminal window:

After deleting the pickle file again, and running the code in a terminal window we see the following interactions:



```
File ToDoFile.pickle was not found.

Do you want to load data from ToDoFile.txt instead? (Y/N) y
Reading data from ToDoFile.txt
Displaying data from ToDoFile.txt

*******************************************
First thing  (High)
Second thing  (High)
Thrid thing  (High)
Fourth thing  (High)
Fifth thing  (High)
Last thing  (High)
*******************************************

Do you want to pickle this data? (Y/N) y
Pickling complete.
```

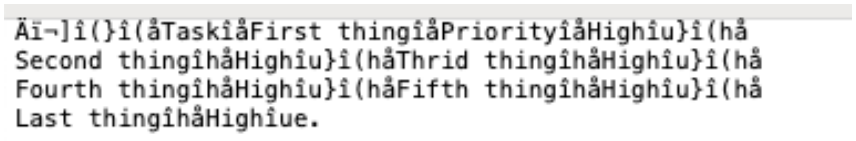
Figure 14: Running without the pickle file



```
Displaying data from ToDoFile.pickle

*******************************************
First thing  (High)
Second thing  (High)
Thrid thing  (High)
Fourth thing  (High)
Fifth thing  (High)
Last thing  (High)
*******************************************

Do you want to pickle this data? (Y/N) n
Data was not pickled.
```

Figure 15: Running with the pickle file

Checking the contents of the new pickle file in a text editor - since we don't see an error message, it appears to be using a different encoding than UTF-8 used in Pycharm, but the text is still quite jumbled.



```
Äï¬]î(}î(åTaskîåFirst thingîåPriorityîåHighîu}î(hå
Second thingîhåHighîu}î(håThrid thingîhåHighîu}î(hå
Fourth thingîhåHighîu}î(håFifth thingîhåHighîu}î(hå
Last thingîhåHighîue.
```

Figure 16: Viewing the contents of ToDoFile.pickle with TextEdit.app

Summary

In this project we created a simple program to demonstrate exception handling and using the pickle module to create/read output files. Within the code we identified a file to be loaded by default when the script runs, then intentionally did not provide that file in an attempt to make the program fail. By using the `try…except` statement, we prevented the program from crashing and allowed it to tell us what we already knew (that the file did not exist) and give us the option of getting data from a different source.

We compared the code used to load a text file, with that to load from a pickle format - and saw that by using this new module, we could read the data into memory without having to know how it's structured in the file.

Likewise, we can output data to a file without having to manually define the structure in the code - yet when we reload the file later, the structure in memory matches that of the originating program. Which can be a more efficient way to transfer data - at least between Python programs.