

Jeff Nelson

June 2, 2020

IT FDN 100 A

Assignment #7

<https://github.com/jnelson22/IntroToProg-Python-Mod07>

Files & Exceptions

Intro

In this paper, I will go over a few interesting and important items to remember that I learned from this week's lecture. Lastly, I will show a program that uses pickling to read and write binary files and an example of error handling.

Week 7 Learnings

This week I learned more about file handling, pickling, and structured error handling. When working with text files you can perform three different types of functions `open()`, `write()`, and `close()`. When the file open function is called you can have it in three different modes, write "w", read "r" or append "a". Python allows for a few different ways to read in data from a file. The first function is `readline()`, which reads one line of data and then moves to the next line.

```
file = open(file_name, "r")
# readline() acts like a "cursor"
data = file.readline() # read one row of data in the file
file.close()
```

Figure 1: Readline example

The next function is `readlines()`, which reads all the lines in a file and returns a list. This is helpful when you want to read in all your data at one time into a list.

```
file = open(file_name, "r")
data = file.readlines() # reads rows of data into a list object
file.close()
```

Figure 2: Readlines example

Using loops is a great way to capture data from a file. The for and while loops can both be used to iterate through the file. For loop has a nice feature that automatically closes the file once done.

```
data = [] # you must initialize the list variable before you use it
for row in open(file_name, 'r'):
    data.append(row) # read one row of data in the file per loop
# automatically closes the file
```

Figure 3: Read File For Loop Example

One other file format is binary. In Python saving to that type is called pickling. Using the pickling function allows you to store data in binary format which obscures the content of the file and can reduce the file's size. It is important to note that the file is not encrypted but rather obscure. See Figure 4 and <https://wiki.python.org/moin/UsingPickle> for an example of pickling in Python.

```
Toggle line numbers
1 # Save a dictionary into a pickle file.
2 import pickle
3
4 favorite_color = { "lion": "yellow", "kitty": "red" }
5
6 pickle.dump( favorite_color, open( "save.p", "wb" ) )

Toggle line numbers
1 # Load the dictionary back from the pickle file.
2 import pickle
3
4 favorite_color = pickle.load( open( "save.p", "rb" ) )
5 # favorite_color is now { "lion": "yellow", "kitty": "red" }
```

Figure 4: Example of Pickle Function

The last topic from this week is structured error handling. In Python, we use try-except to trap errors that the user has caused when using the program. This is good to add to your code whenever you think there might be a human cause error.

```
try:
    quotient = 5/0
    print(quotient)
except:
    print("There was an error! <<< Custom Message!\n")
```

Figure 5: Example of Try-Except Error Handling

The exception class in Python allows us as programmers the ability to add in custom error messages that make more sense to them than the developer. You can also create your own custom exception classes.

```
class CustomError(Exception):
    """ Some custom error info in the DocString """
    def __str__(self):
        return 'Some custom error message'
```

Figure 6: Example of Custom Error Message Class

Adding on to To-do List Python Program

This program's problem statement was to create a program that used pickling and structured error handling. This allowed for a different file interface and more user-friendly error messages.

First, I took my code for last week and modified it to include reading for a binary file using the `pickle.load()` function.

```
file = open(file_name, "rb")
list_of_rows = pickle.load(file)
file.close()
return list_of_rows # 'Success'
```

Figure 7: Pickle Function Reading from a File

Next, the update I made was to the file writing function. Writing to a binary file I used the `pickle.dump()` function.

```
objFile = open(file_name, "wb")
pickle.dump(list_of_rows, objFile)
objFile.close()
print("Tasks saved to ToDoList.dat")
return list_of_rows # 'Success'
```

Figure 8: Pickle Function Writing to a File

Here is an example of what is in the binary once it is read into the programs.

```
***** The current Tasks ToDo are: *****
Water (Hi)
Clean (Hi)
Dishes (Low)
*****
```

Figure 9: Example of Data Read into Memory from a Binary in PyCharm

Here is what that binary looks like when opened in a text editor.

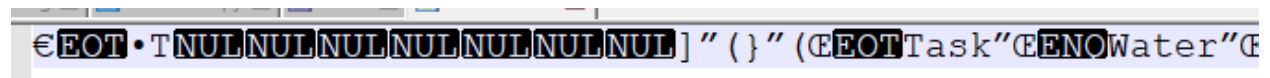


Figure 10: Screenshot of the Same Data Opened in a Text Editor

Last was to add some file error handling. I wanted to make sure the user saw a helpful message when missing the ToDoList.dat file.

```
try:
    lstTable = Processor.read_data_from_file(strFileName) # read file data
except FileNotFoundError as e:
    print("ToDoList.dat not found. Please create!")
    sys.exit()
except Exception as e:
    print("There was a non-specific error!")
    print(e)
    sys.exit()
```

Figure 11: Screenshot of Error Handling Code

Here is the error message they receive in PyCharm.

```
ToDoList.dat not found. Please create!
```

Figure 12: Screenshot of the Error Output in PyCharm

Here is the error message again but in the command prompt.

```
C:\> Command Prompt
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\jnelson>python "C:\_PythonClass\Assignment07\Assignment07.py"
ToDoList.dat not found. Please create!
```

Figure 13: Screenshot of the same Error Message in the Command Prompt

Summary

In summary, I learned more about file handling, pickling, structured error handling, and GitHub markdown language. I went through the process of how I updated my code to use binary files and better error handling. I found this assignment relatively easy but I found that creating the GitHub page pretty time-consuming. Though it was time-consuming I found it useful to know. Overall, I'm happy with the way this assignment and webpage turned out.