Steven Simpers
May 30, 2023
Foundations of Programming: Python
Assignment 07
https://github.com/ssimpers/IntroToProg-Python-Mod07.git (External Site)

Pickling and Structured Error Handling

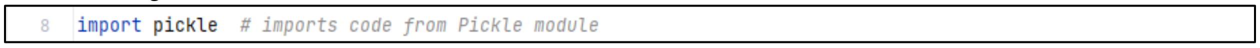
Introduction

This week's lesson showed methods for reading and writing to files, and how built-in functions can be wrapped within custom functions to make them more user friendly. Structured error handling was introduced to better communicate errors to a user or to preemptively address errors. The assignment problem statement is to create a script demonstrating pickling and structured error handling.

Pickle Module

The first statement in the script, shown in Listing 1, imports code from the Pickle module that allows the script to interact with binary files.

Listing 1: Pickle module

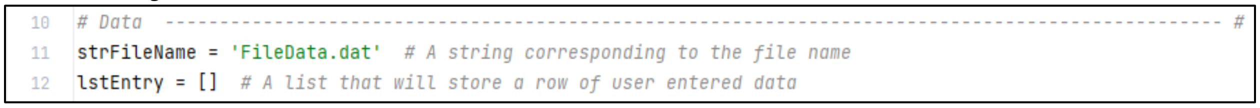


```python
import pickle  # imports code from Pickle module
```

Data

The script then defines variables and constants in Listing 2. The name of the file that will store the list data is assigned to strFileName and lstEntry is defined as an empty list to store user input information.

Listing 2: Data

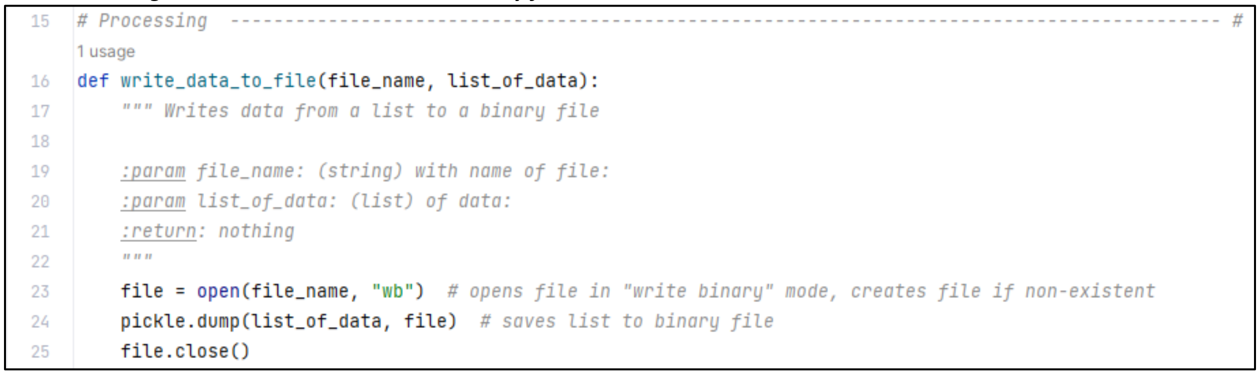


```python
# Data  ---------------------------------------------------------------------------------------------- #
strFileName = 'FileData.dat'  # A string corresponding to the file name
lstEntry = []  # A list that will store a row of user entered data
```

Processing – Writing Data to a File

Listing 3 shows a custom function that opens a file in "write binary" mode and saves a list to the file using the pickle module's "dump" function. The file is then closed and the custom function returns nothing.

Listing 3: Function to write data to a binary file

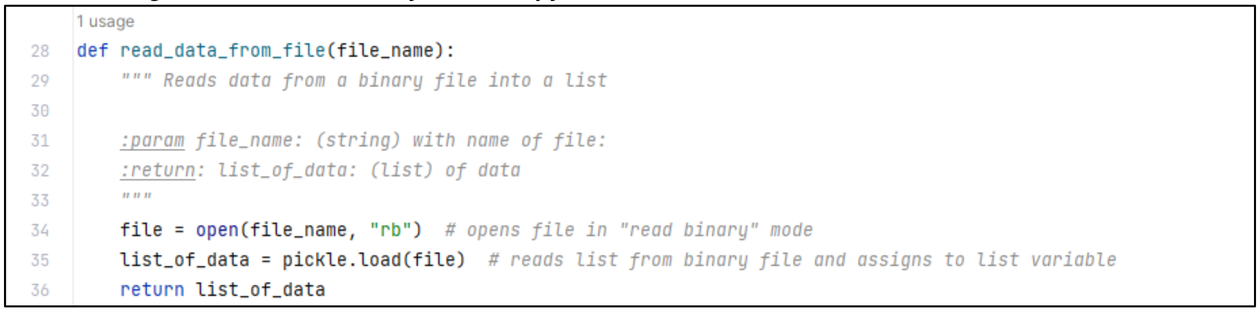


```python
# Processing  ----------------------------------------------------------------------------------- #
1 usage
def write_data_to_file(file_name, list_of_data):
    """ Writes data from a list to a binary file

    :param file_name: (string) with name of file:
    :param list_of_data: (list) of data:
    :return: nothing
    """
    file = open(file_name, "wb")  # opens file in "write binary" mode, creates file if non-existent
    pickle.dump(list_of_data, file)  # saves list to binary file
    file.close()
```

Processing – Reading Data from a File

The next function in Listing 4 receives the file name and opens the file in “read binary” mode. It uses the pickle module’s “load” function to store the data in a list that is returned from the custom function.

Listing 4: Function to read data from a binary file

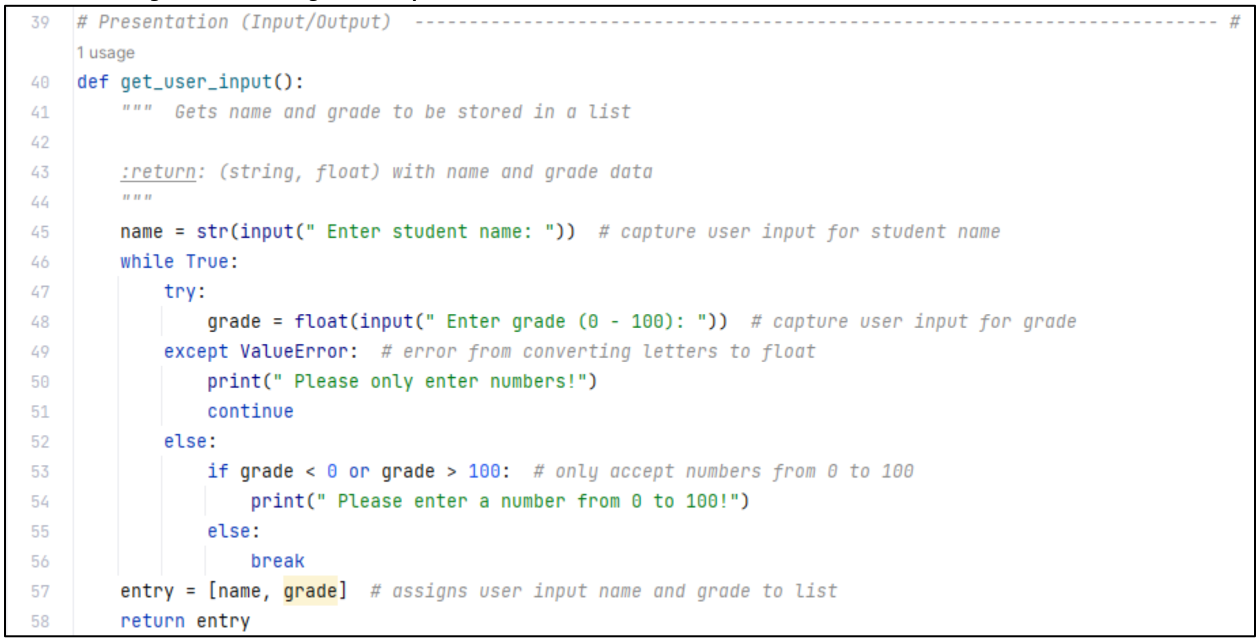


```python
1 usage
def read_data_from_file(file_name):
    """ Reads data from a binary file into a list

    :param file_name: (string) with name of file:
    :return: list_of_data: (list) of data
    """
    file = open(file_name, "rb")  # opens file in "read binary" mode
    list_of_data = pickle.load(file)  # reads list from binary file and assigns to list variable
    return list_of_data
```

Input/Output – Get User Inputs

The next function in Listing 5 doesn’t accept parameters however it prompts the user for information and returns it in a list. The input function prompts the user for a student name and converts it to a string. A while loop was created to contain structured error handling so that if errors occur when converting the user input grade to a float value, then the user is notified and asked to enter another response. A likely error to occur is the “ValueError” which would result from trying to convert a string of letters to a float value instead of converting a string of numbers representing the student’s grade. An “if” statement is additionally used to check if the number is between 0 and 100 and ask the user to re-enter a value if it is not. Otherwise, the program breaks out of the while look and returns a list of the student’s name and grade.

Listing 5: Function to get user inputs

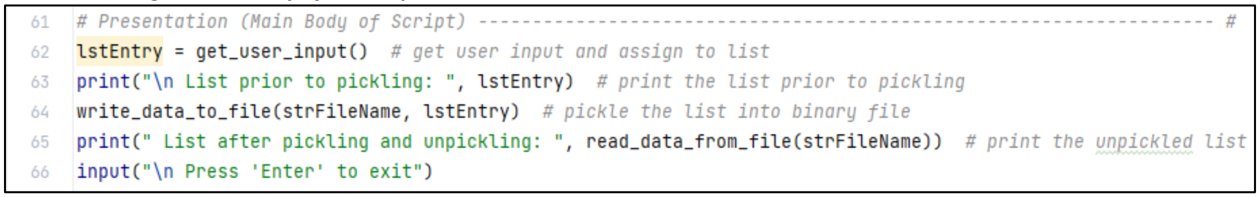


```
# Presentation (Input/Output)  ------------------------------------------------------------------------- #
1 usage
def get_user_input():
    """  Gets name and grade to be stored in a list

    :return: (string, float) with name and grade data
    """
    name = str(input(" Enter student name: "))  # capture user input for student name
    while True:
        try:
            grade = float(input(" Enter grade (0 - 100): "))  # capture user input for grade
        except ValueError:  # error from converting letters to float
            print(" Please only enter numbers!")
            continue
        else:
            if grade < 0 or grade > 100:  # only accept numbers from 0 to 100
                print(" Please enter a number from 0 to 100!")
            else:
                break
    entry = [name, grade]  # assigns user input name and grade to list
    return entry
```

Main Body of the Script

The main body of the script in Listing 6 executes statements that utilize the custom functions discussed above and prints information back to the user. User inputs are captured from the "get_user_input" function and are assigned to a list. The list is printed to the user as a record of the information before pickling. The list is then written to a binary file (pickled) using the "write_data_to_file" function on line 64. The list from the binary file is then read (unpickled) and printed back to the user for comparison to the before pickling list. The input function is used on line 66 to allow the user to read the information before the script ends.

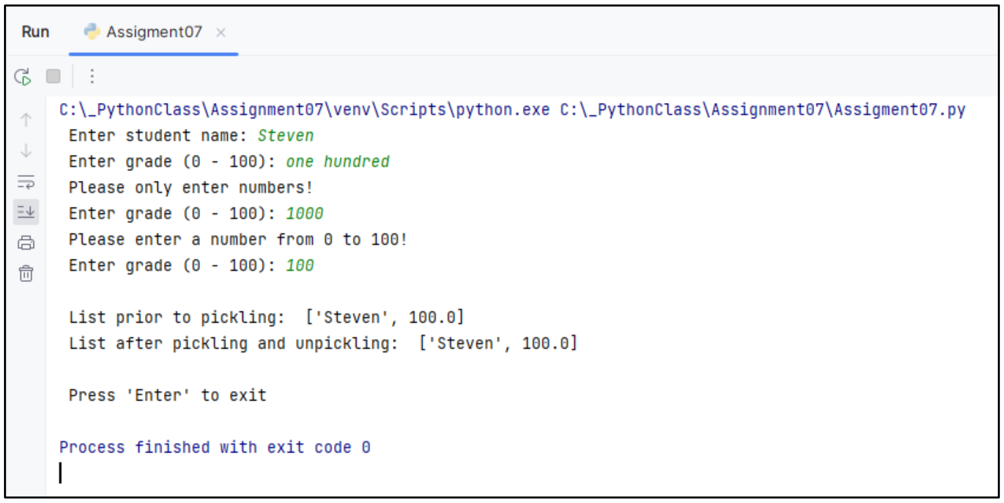
Listing 6: Main body of the script



```
61  # Presentation (Main Body of Script) ------------------------------------------------------------ #
62  lstEntry = get_user_input()  # get user input and assign to list
63  print("\n List prior to pickling: ", lstEntry)  # print the list prior to pickling
64  write_data_to_file(strFileName, lstEntry)  # pickle the list into binary file
65  print(" List after pickling and unpickling: ", read_data_from_file(strFileName))  # print the unpickled list
66  input("\n Press 'Enter' to exit")
```

Results

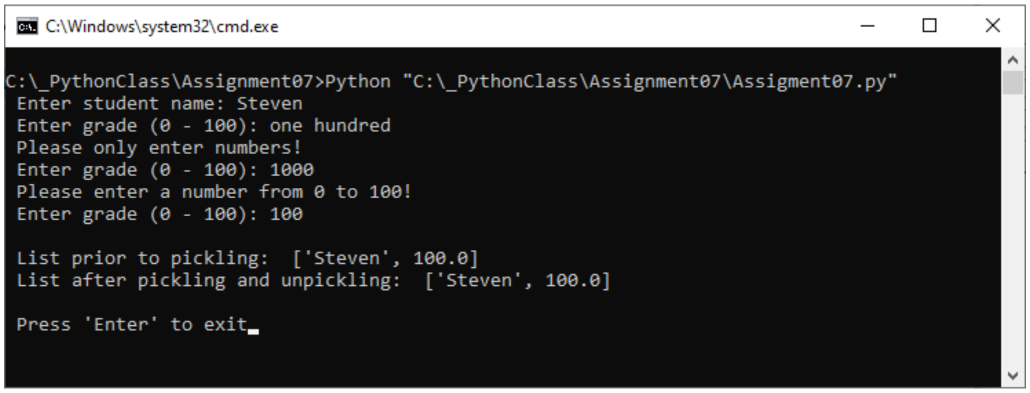
An example of the assignment script running in PyCharm is provided below in Figure 1. The structured error handling catches the string "one hundred" entered as the grade when it tries to convert it to a floating value. Additional conditional statements ask the user for a value between 0 and 100. The information is shown in a list before and after pickling which appears the same.



```
C:\_PythonClass\Assignment07\venv\Scripts\python.exe C:\_PythonClass\Assignment07\Assigment07.py
 Enter student name: Steven
 Enter grade (0 - 100): one hundred
 Please only enter numbers!
 Enter grade (0 - 100): 1000
 Please enter a number from 0 to 100!
 Enter grade (0 - 100): 100

 List prior to pickling:  ['Steven', 100.0]
 List after pickling and unpickling:  ['Steven', 100.0]

 Press 'Enter' to exit

Process finished with exit code 0
```

Figure 1: Python script running in PyCharm

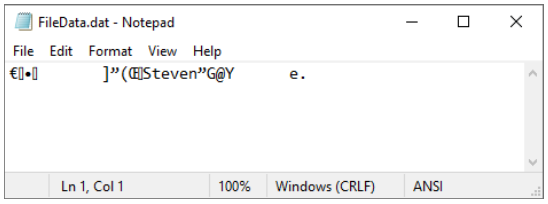
Figure 2 below shows the script running similarly in the command window.



```
C:\_PythonClass\Assignment07>Python "C:\_PythonClass\Assignment07\Assigment07.py"
 Enter student name: Steven
 Enter grade (0 - 100): one hundred
 Please only enter numbers!
 Enter grade (0 - 100): 1000
 Please enter a number from 0 to 100!
 Enter grade (0 - 100): 100

 List prior to pickling:  ['Steven', 100.0]
 List after pickling and unpickling:  ['Steven', 100.0]

 Press 'Enter' to exit_
```

Figure 2:Python script running in Command Window

Figure 3 shows the contents of the binary file. The information is obfuscated but not encrypted.



```
€•       ]”(ŒSteven”G@Y      e.
```

Figure 3: Data Saved in “FileData.dat” File

Summary

The assignment tested our knowledge of pickling and structured error handling. The assignment also provided hands-on experience with Markdown language while replicating the assignment document in GitHub.