Ryan Lupinski
02.23.2022
Foundations of Programming: Python
Assignment 07
Github URL: https://github.com/ryanlupinski/IntroToProg-Python-Mod07

Pickling & Structured Error Handling

1 Introduction

This write-up will discuss the concept of structured error handling as well as pickling. A Python script will be created that tracks a work log by ID and work performed. The script will also take in the time stamp using functionality built into the datetime module. When work is entered, a timestamp is recorded. Structured erroring handling will be integrated to ensure the user is guided through the input process smoothly and plain English error messages are returned to the console.

2 Pickling

Python has a built-in module called pickle that is used to serialize and de-serialize object data. "What pickle does is that it 'serializes' the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script." (Geeks for Geeks, https://www.geeksforgeeks.org/understanding-python-pickling-example/, Feb 2022) (External Site). Pickling can also abstract the data you want to store on the disk and reduce the size of the file to save space. The work log is stored on the disk as a binary file named WorkLog.dat. See Figure 1. The work log is obscured in binary and needs to be unpickled and printed to the console to be reable.

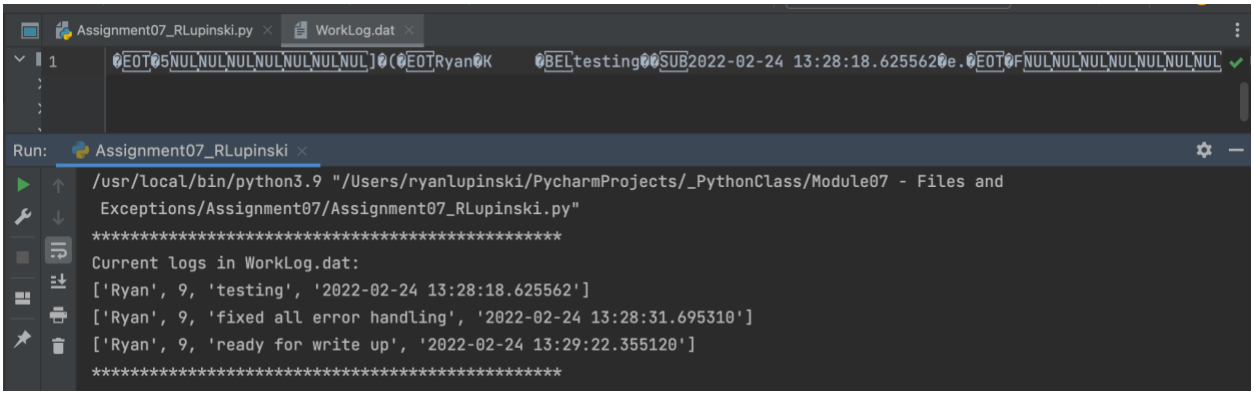

Figure 1. WorkLog.dat binary file, and plain text unpickled entries in Pycharm

Exploring the first section of code, the functionality the program needs such as time stamps and serial/de-serialization must be imported from the corresponding modules using the import command. See Listing 1.

```
# Import Modules ------------------------------------------------------------ #
import pickle  # import pickle module for binary handling
from datetime import datetime  # import datetime module

# Data ---------------------------------------------------------------------- #
# Declare variables and constants
strFileName = "WorkLog.dat"  # work log binary file
intID = ""  # initialize ID string
strWork = ""  # initialize work log string
lstWorkLog = []  # initialize list of work logs
```
Listing 1 Importing modules and initializing variables

The module datetime is imported for the datetime library. Now that we have pickling available, the binary data in the .dat file can be deserialized and pulled into the console.

2.1 Reading and saving data with Pickle

The first function that is save_data_to_file(), that takes in a file name and a list of work to be saved to the .dat file. See listing 2. A connection with the file is opened in append-binary ‘ab’ mode. This allows for the dile to be created in the event it does not exist. Using the module pickle and the function dump, pickle.dump, accepts arguments of the list of work and the objectfile and saves the data. Finally the connection is closed.

```
# Processing  --------------------------------------------------------------- #
class Processor:
    @staticmethod
    def save_data_to_file(file_name, list_of_data):
        """
        saves data to the binary file
        :param file_name: name of file to save data
        :param list_of_data: list of work logs
        :return:
        """
```

```
        objFile = open(file_name, "ab")  # open dat file in append mode
binary
        pickle.dump(list_of_data, objFile)  # write work log to file
        objFile.close()  # close connection to file
```

Listing 2 Saving data to a binary file with pickle.dump

Next, reading data from the .dat is defined in IO class using function read_data_from_file() that accepts the file name as an argument. See Listing 3. When the function is called in the main body the file name is passed in and a connection is opened in read-binary 'rb' mode. A while loop uses try-except to print each line of the file using pickle.load. When the end of the file is attempted to be loaded, the EOFError causes the except statement to execute and the loops breaks. The connection to the file is closed and the user can see all the work logs in the console.

```
# Presentation (Input/Output)  -------------------------------------------- #
class IO:
    @staticmethod
    def read_data_from_file(file_name):
        """
        reads data from the binary file
        :param file_name: name of file to read data
        :return:
        """
        print("*********************************************")
        print(f"Current logs in {file_name}:")  # prints current data from
WorkLog.dat
        objFile = open(file_name, "rb")  # open connection to file in read
mode binary
        while True:  # loop through all objects until end of file error
occurs
            try:
                data = pickle.load(objFile)  # load data from file to data
var
                print(data)  # print all the data to the console
            except EOFError:  # when end of file error occurs, break the loop
                break
        objFile.close()  # close connection to file
        print("*********************************************")
        print("\n")
        return
```

Listing 3 Reading data from a binary file with pickle.load

2.2 Main Body of Script

Now that the modules are imported, the variables are initialized, and the read (unpickle) and write (pickle) functions are defined in the IO and Processor classes, a while loop will presenter the user options to log work or exit the program. The current work log is always presented using at the beginning before prompting the user to type 1 as n input to add a log or any key to exit the program. If 1 is selected, the script enters the if statement and a try-except error handler. This ensures the user enters logs correctly. See listing 4 below.

```
# Main Body of Script  ---------------------------------------------------- #
```

```
while True:  # loop through option to user
    IO.read_data_from_file(strFileName)  # display current work log
    choice = input("Type 1 to log work, or hit any key to exit: ")  # option
to add log or exit
    if choice == "1":  # if choice == 1 add a new log
        try:  # trying code for errors
            strName = input("Enter your name: ")  # prompt user for name
            if strName.isnumeric():  # if numeric entry is found
                raise WorkLogNameError()  # raise custom error
            intID = int(input("Enter an ID: "))  # get worker's ID
            strWork = str(input("Log work: "))  # get string of work logged
            ts = str(datetime.now())  # get timestamp, convert to string
            lstWorkLog = [strName, intID, strWork, ts]  # list of
name,ID,work logged, and timestamp
            Processor.save_data_to_file(strFileName, lstWorkLog)  # pass file
and work log list to function
        except ValueError as e:  # raise exception for ID non-numeric
            print("ID must be a number")
            print("Built-In Python error info: ")
            print(e, e.__doc__, type(e), sep='\n')  # return python error
docs
        except Exception as e:  # generic catch-all error exception
            print("There was a non-specific error!")
            print("Built-In Python error info: ")
            print(e, e.__doc__, type(e), sep='\n')  # return python error
docs
    else:  # else break the loop and end the program
        break
```

Listing 4 Main Body of Script

3 Structured Error Handling

3.1 Try-except

The program uses the try-except structure when taking input from the user. The built-in ValueError is part of Python's code and when the user attempts to enter a non-numeric ID the try section will cause this specific error and will jump to that specific exception statement. It will then print the error name, a short description, and return the user to the beginning of the while loop to try again. If an error occurs that is not the ValueError, the final except Exception statement will execute as a catch-all, returning relevant info and prompting the user to try again.

3.2 Custom error handling

A custom error function was created to showcase raising custom exception. If the user types a number for their name, technically any number can be a string when it is passed through the input function. The program wants to ensure that only alpha-numeric entries are accepted, so strName.isnumeric() is written in an if statement to raise the custom exception WorkLogNameError. See listing 5.

```
# Custom Exception Class  ---------------------------------------------- #
class WorkLogNameError(Exception):
    """Can not have numeric entry for name"""

    @staticmethod
```

```
    def __int__(self):  # custom error handling example for when a non-
alphanumeric name is tried
        return 'Can not have numeric entry for name'
```

4 Summary

Pickling is a tool to serialize and deserialize binary data stored on the disk to readable and modifiable objects in memory. The pickle.load() and pickle.dump() functions are the two basic methods for handling binary data. Structured error handling is an important part of writing customer facing code, in which the user may not have technical programming knowledge. This type of programming allows for fluid use of programs and makes preemptively designing for errors easy.