Matt Dunn
August 18, 2023
IT FDN 110 A
Assignment 07
https://github.com/madunn5/IntroToProg-Python-Mod07

Pickling in Python

Introduction

In this paper I will go over how I created a Python program that implements pickling and exception handling into the code. This was my first opportunity to test what I have learned so far and make a program from scratch in whatever way I see fit. It also gave me the opportunity to research pickling and exception handling as that was not something that was covered very deeply in the lecture purposefully.

Researching Pickling in Python

Pickling in Python means to serialize and deserialize data types, most commonly be turning the loaded data in binary data. Pickle is the name of the Python specific module that is used to accomplish serializing/deserializing the data. Pickle.dump() or Pickle.dumps() are the functions used to serialize an object, the difference being that .dump() writes the data to a file and .dumps() represents it as a byte object. Pickle.load() or Pickle.loads() are the functions used to deserialize an object, and just like the dump functions the difference between the two is the same – either from a file or from byte object. For this assignment, I used .dump() and .load(). There will be examples of both functions later in this assignment.

The source I used to research Pickling was https://www.datacamp.com/tutorial/pickle-python-tutorial and I found their examples to be very helpful as I started writing my code.

Researching Exception Handling in Python

Exception Handling in Python must deal with how errors in the program can arise, and how the code handles them. So far in the course, I have had instances where I've had to code around errors, but they have mostly been handled by use if/else statements. Exception Handling allows for the use of built in functions that can relay the error to the user, but they also allow for custom error creating as well. These functions can range from ValueErrors, which occur when the wrong argument is given, or errors like ZeroDivisionError that are clearer in their name. I found that using Exception as e: is the type of handling that worked best for me in the code as it is the broadest and allows for catching more errors that I might not be thinking of in most situations. In situations where one type of error is most common, it would make more sense to use the specific type of Error, like a ZeroDivisionError for a math function where dividing by 0 is possible. For this program I felt like taking a broader approach would be preferable.

The sources I used to research Exception Handling were https://www.geeksforgeeks.org/python-try-except/ , https://www.tutorialspoint.com/python3/python_exceptions.htm#, https://embeddedinventor.com/python-except-exception-as-e-meaning-explained/. I found each website to be helpful for all the examples I ended up using in my code.

Creating & Testing the Program

Once I had completed enough research to feel confident about performing the task at hand, I started to right my code one section at a time. I built off our previous homework assignments up to this point, and I thought that trying to build something like Assignment 06 might be a good challenge for me, so that was the path I started down.

The first section of the code acts a log of information about what the code should accomplish, as well as any changes made to the code so that anyone can come in and get an idea of what is going on.

```
# ------------------------------------------------ #
# Title: Assignment 07 - Pickling
# Description: A simple example of storing data in a binary file
#              and then using pickle to unpickle it. Additionally,
#              the program can print the data to a text file,
#              delete all previously saved data and show the user
#              the binary data
# ChangeLog: (Who, When, What)
# MDunn,8/17/2023,Created Script
# ------------------------------------------------ #
```

Figure 1. Top of the code where detail can be added about what the code should accomplish and any changes that have been made to it.

The next part of the code is the declaration of variables, as well as importing pickle to the program so that we can use the functions associated with it.

```
import pickle

# Data ------------------------------------------ #
user_id = ''  # Captures the user ID input
user_name = ''  # Captures the username input
user_data = []  # A list that acts as a table of rows
filename_pickle = 'data.pkl'  # The name of the binary file used for pickling data
filename_txt = 'data.txt'  # The name of the text file that will show the data in plain english
choiceStr = ''  # Captures the choice made from the option menu
data_list = []  # A list that acts as the data currently in the program
```

Figure 2. Declaring variables in their own section to have an overview of what to expect down below in the actual code and importing pickle.

Now we enter the “Processing” part of the code, where I created two Classes, Processor and InputOutput. The Processor Class holds all the functions that have to do with processing data, and the InputOutput Class holds everything that deals with the inputs and outputs of the code. I’ll start with the functions in the Processor Class.

The first function is save_pickled_data_to_file. This function takes two parameters, data and filename. This function will serve as the save function and will save the pickled data to a specific filename, in this case it should be filename_pickle that was outlined in the “data” section of the code. Here I have my first

instances with the pickle.dump() function, as well as my first Exception as e: function. Pickle.dump() will serialize the data and save it to the specified file, and all of this is built within a try/except function that will print any errors that occur to the user's screen.

```
28      @staticmethod
29      def save_pickled_data_to_file(data, filename):
30          """ Saves current data using pickle.dump()
31
32                  :param data: (string) with data to save
33                  :param filename: (string) with name of file
34                  :return: nothing
35                  """
36          try:
37              with open(filename, 'ab') as file:
38                  pickle.dump(data, file)
39              print()  # space for formatting
40              print(f"Data saved to {filename} successfully.")
41          except Exception as e:
42              print(f"Error while saving data: {e}")
```

Figure 3. Defining the function save_pickled_data_to_file.

The next function, save_unpickled_data_to_file, does something very similar to the previous function, except this time it's saving the deserialized data to a text file. I thought this would be a good function include in my code should the user wish to have a text file copy of whatever their data is so that they can refer to it outside of the code window, if needed. For this function to work as expected, the data param will need to be saved to the local data variable. More on this later.

```
    @staticmethod
    def save_unpickled_data_to_file(data, filename):
        """ Saves current data to text file. Because we are using read_pickled_data_from_file in order
        to create the data_list variable, the data is automatically printed to the screen, allowing the
        user to see what is being saved to the text file

                :param data: (string) with data to save
                :param filename: (string) with name of file
                :return: nothing
                """
        try:
            with open(filename, 'w') as file:
                if data:
                    file.write('user_id,user_name\n')  # Adds user_id and user_name as a header for each column
                    for row in data:
                        file.write(row[0] + ',' + row[1] + '\n')
                    print()  # space for formatting
                    print(f'The above data has been saved to {filename}!\n')
                else:
                    print('No data to save.\n')
        except Exception as e:
            print(f"Error while saving data: {e}")
```

Figure 4. Defining save_unpickled_data_to_file.

The next function is read_pickled_data_from_file. This is very similar to save_unpickled_data_to_file, except that instead of saving to a text file it will print to user's window. This will be useful for instances when the user doesn't necessarily want to print the data to text (perhaps it's sensitive data), but they still would like to view it so they can view it within the program. This function as employees a except EOFError. This error is an "end-of-line" error and is especially useful when paired with a while True loop, which is exactly what I have below. This exception acts a way to break the While loop once the loop has run out of data to loop through.

```
    @staticmethod
    def read_pickled_data_from_file(filename):
        """ Reads pickled data from a file and returns it in plain english

                :param filename: (string) with name of file
                :return: (list) of data read from the file
                """
        try:
            with open(filename, 'rb') as file:
                data_list = []
                while True:
                    try:
                        user_data = pickle.load(file)
                        data_list.append(user_data)
                    except EOFError:  # this is an end-of-line error which triggers the break in the loop
                        break
                if data_list:  # if the data_list exists, this loop will run
                    for row in data_list:
                        print(row[0], row[1], sep=',')
                else:
                    print('No data is available.')
                return data_list
        except Exception as e:
            print(f"Error while reading data: {e}")
            return None
```

Figure 5. Defining read_pickled_data_from_file.

The next function is delete_pickled_data_from file. This function acts as a way for the user to delete any previously saved data and essentially just opens the file in write mode before passing and not doing anything.

```
93      @staticmethod
94      def delete_pickled_data_from_file(filename):
95          """ Deletes the previously saved data
96
97                  :param filename: (string) with name of file
98                  :return: nothing
99                  """
100         try:
101             with open(filename, 'wb'):
102                 pass
103             print(f"All data has been deleted from {filename} successfully.")
104         except Exception as e:
105             print(f"Error while saving data: {e}")
```

Figure 6. Defining delete_pickled_data_from_file.

The final function of the Processor Class is read_pickled_data_as_binary. This function I built because I thought user might find it find to look at their serialized data. It essentially just prints the binary data to the screen after the reading the file.

```
107     @staticmethod
108     def read_pickled_data_as_binary(filename):
109         """ Returns pickled data in binary to the user, so they can see what it looks like
110
111                 :param filename: (string) with name of file
112                 :return: (bytes) of binary data read from the file
113                 """
114         try:
115             with open(filename, 'rb') as file:
116                 binary_data = file.read()
117                 print('Below is what your data looks like in binary!', '\n')
118                 print(binary_data)
119         except Exception as e:
120             print(f"Error while reading data: {e}")
121             return None
```

Figure 7. Defining read_pickled_data_as_binary.

The InputOutput Class is next, which represents all the input and output tasks required for this code. These functions in general are a little less challenging when it comes to the coding, but they are arguably just as important (if not the most important) because these inputs/outputs are involved in all the Processor Class functions as well.

The first function of the InputOutput Class is output_menu_choices, which will print the menu of options shown to the user at the beginning of each loop. This function has no parameters and simply acts as a print statement of the menu.

```
127     @staticmethod
128     def output_menu_choices():
129         """  Display a menu of choices to the user
130 
131         :return: nothing
132         """
133         print('''
134             Menu of Options
135             1) Show Current Data
136             2) Add New Data
137             3) Create a New File/Delete Existing Data
138             4) Show Binary Data
139             5) Save Data to Text File
140             6) Exit Program
141             ''')
142         print()  # Add an extra line for looks
```

Figure 8. Defining output_menu_choices.

The next function, input_menu_choice, captures the user's input for what menu choice they would to make.

```
144     @staticmethod
145     def input_menu_choice():
146         """ Gets the menu choice from a user
147 
148         :return: string
149         """
150         choice = str(input("Which option would you like to perform? [1 to 6] - ")).strip()
151         print()  # Add an extra line for looks
152         return choice
```

Figure 9. Defining input_menu_choice.

The final function is input_new_user_id_and_user_name, which takes the user's inputs for user_id and user_name and returns them. In this function I employ a couple of if/else statements with while True loops in order to make sure that the user gives the correct input format. The reason I opted for this instead of Exception Handling was that I found it easier to navigate how to control specific criteria by using if/else.

```
154     @staticmethod
155     def input_new_user_id_and_user_name():
156         """  Gets user_id and user_name values to be added to the list
157 
158         :return: (string, string) with user_id and user_name
159         """
160         # if the input is not a number, the code will repeat until a valid input is received
161         while True:
162             user_id = input('Enter an ID: ')
163             if user_id.isnumeric():
164                 break
165             else:
166                 print('Please only use numbers for the User\'s ID!')
167 
168         # if the user_name is blank the code will ask for a valid input
169         while True:
170             user_name = input("Enter a name: ")
171             if user_name.strip():
172                 break
173             else:
174                 print('User name cannot be blank')
175         user_data = [user_id, user_name]
176         return user_data
```

Figure 10. Defining input_new_user_id_and_user_name.

Now that all the Classes and their subsequent functions have been created and defined, it's time to work them into the main body of the script! You'll notice that for choices 2 and 5 the data param is defined before running the function. This is so that the data is loaded into local memory so that the function uses the correct data. The other options either run only with a filename param (which is a global variable defined at the top of the code) or don't require a function to execute. All of this is done under a while True loop so that the menu of choices is always presented to the user until choice 6 is selected.

```
# Presentation ---------------------------------- #
while True:
    InputOutput.output_menu_choices()  # Shows menu of options
    choice_str = InputOutput.input_menu_choice()  # Get menu option

    if choice_str.strip() == '1':  # View current data
        print('user_id,user_name')
        Processor.read_pickled_data_from_file(filename=filename_pickle)
        continue  # to show the menu

    elif choice_str == '2':  # Add and save data
        user_data = InputOutput.input_new_user_id_and_user_name()
        Processor.save_pickled_data_to_file(data=user_data, filename=filename_pickle)
        continue  # to show the menu

    elif choice_str == '3':  # Overwrite the file, starts with user_id and user_name at top for formatting
        Processor.delete_pickled_data_from_file(filename=filename_pickle)
        continue  # to show the menu

    elif choice_str == '4':  # View the binary data
        Processor.read_pickled_data_as_binary(filename=filename_pickle)
        continue

    elif choice_str == '5':  # Add and save data
        data_list = Processor.read_pickled_data_from_file(filename=filename_pickle)
        Processor.save_unpickled_data_to_file(data=data_list, filename=filename_txt)
        continue

    elif choice_str == '6':  # Exit Program
        print("Goodbye!")
        break  # by exiting loop

    else:
        print('Please choose a number between 1 and 6!')
```

Figure 11. Main body of the script incorporating all the functions.

Last step now is to test the code! I started from scratch, so there was currently no data saved when I started the test. My goal is to add two users, John Doe and Jane Doe, to the script and then view my current data as well as view the binary data before saving to a text file.

```
/Users/Matt/Assignment07/bin/python /Users/Matt/Documents/_PythonClass/Assignment07/Assignment07.py

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 1

user_id,user_name
Error while reading data: [Errno 2] No such file or directory: 'data.pkl'

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 2

Enter an ID: 1
Enter a name: John Doe

Data saved to data.pkl successfully.
```

Figure 12. Running the program in PyCharm to test. The file doesn't currently exist at the beginning of the test, so exception is thrown. I then add John Doe successfully.

```
            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 2

Enter an ID: 2
Enter a name: Jane Doe

Data saved to data.pkl successfully.

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 1

user_id,user_name
1,John Doe
2,Jane Doe
```

Figure 13. Add Jane Doe successfully and then view current data successfully.

```
            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 4

Below is what your data looks like in binary!

b'\x80\x04\x95\x14\x00\x00\x00\x00\x00\x00\x00]\x94(\x8c\x011\x94\x8c\x08John Doe\x94e.\x80\x04\x95\x14\x00\x00\x00\x00\x00\x00\x00]\x94(\x8c\x012\x94\x8c\x08Jane Doe\x94e.'

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 5

1,John Doe
2,Jane Doe

The above data has been saved to data.txt!
```

Figure 14. Successfully viewed the binary data before saving to a text file.

```
            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 6

Goodbye!

Process finished with exit code 0
```

Figure 15. Exited the program successfully.

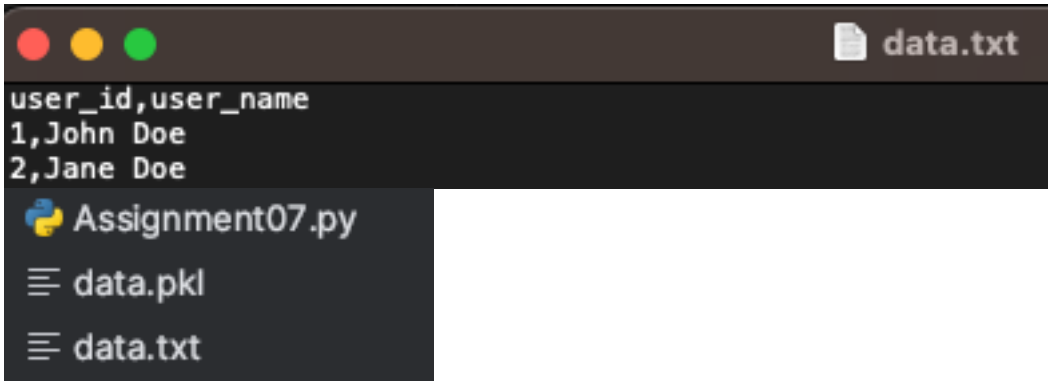

Figure 16. Text file after being saved, as well as the serialized file saved in the same folder.

Running the Program in the Terminal

The last thing to test for this assignment was to the run the program in the Mac Terminal. I'm going to delete the previous file and save the names Ben Johnson and Molly Johnson now that I know their names.

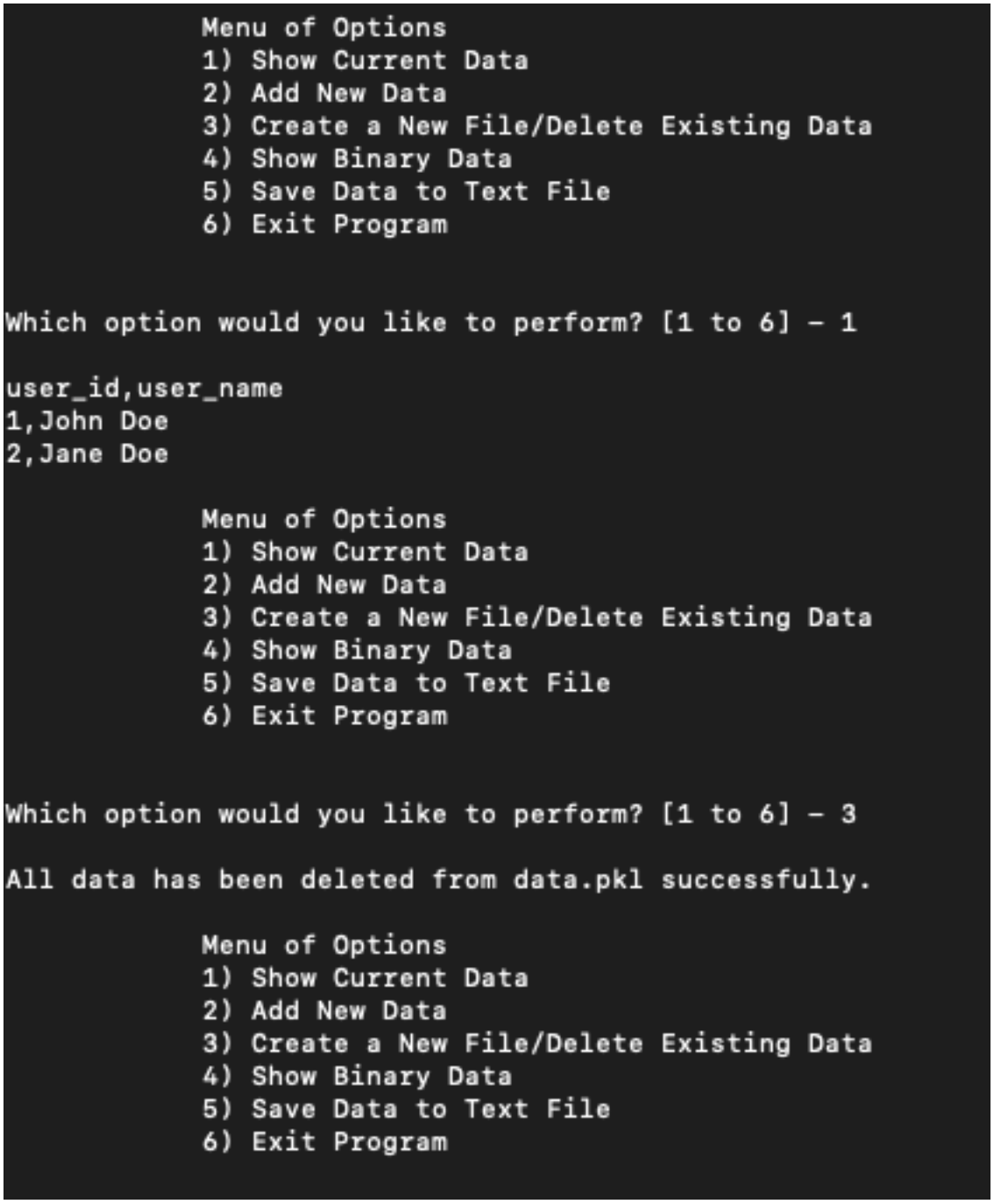

Figure 17. Deleting the previously saved data.

```
Which option would you like to perform? [1 to 6] - 1

user_id,user_name
No data is available.

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 2

Enter an ID: 1
Enter a name: Ben Johnson

Data saved to data.pkl successfully.

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 2

Enter an ID: 2
Enter a name: Molly Johnson

Data saved to data.pkl successfully.
```

Figure 18. Confirming that the previous data was deleted before adding in the new data.

```
            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 1

user_id,user_name
1,Ben Johnson
2,Molly Johnson

            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 5

1,Ben Johnson
2,Molly Johnson

The above data has been saved to data.txt!


            Menu of Options
            1) Show Current Data
            2) Add New Data
            3) Create a New File/Delete Existing Data
            4) Show Binary Data
            5) Save Data to Text File
            6) Exit Program


Which option would you like to perform? [1 to 6] - 6

Goodbye!
```

Figure 19. Confirming data was saved before saving to text file.

Figure 20. Updated text file.

Summary

This assignment was a ton of fun to be able to create program from scratch with this much functionality. I've really enjoyed the progress that I've made, and I can't wait for the next assignment!