Lauren Ferrier
November 30, 2022
Foundations of Programming, Python
Assignment 07
Git Hub Repository
Git Hub Website

Managing Rabbits - A Python Script of Pickling & Error handling

Introduction

This document details the steps taken to create a script for managing our user's love for all things rabbits with a rabbit management list! The theme with this script is similar to previous modules that implemented "todo" and home inventory lists with loops and lists, but this week's script has instead been designed to demonstrate pickling and error handling.

Steps Taken for Assignment 07

1. This assignment was completed within PyCharm (Jet Brains, 2022).

2. Within PyCharm, I opened the Lab7-1_Starter.py that was provided with this week's course materials. Our module notes this week (_MOD7PYTHONPROGRAMMINGNOTES.DOCX 2022, PG 14) contained an exercise that got us practicing pickling.

```
"
import pickle  # This imports code from another code file!

# Data -------------------------------------------- #
strFileName = 'AppData.dat'
lstCustomer = []

# Processing -------------------------------------- #
def save_data_to_file(file_name, list_of_data):
    pass  # TODO: Add code here

def read_data_from_file(file_name):
    pass  # TODO: Add code here

# Presentation ------------------------------------ #
# TODO: Get ID and NAME From user, then store it in a list object
# TODO: store the list object into a binary file
# TODO: Read the data from the file into a new list object and display the
contents
```

*Figure 1 Lab 7-1 served as the initial inspiration behind our eventual, completed script. Note the use of **import pickle** to signify what functions we will need to complete under **save_data_to_file** and **read_data_from_file** functions.*

3. Our notes had previously provided us a further example on pickling functions that I reviewed to help me begin filling out the TODO portions of our started script above.

```
# Now we store the data with the pickle.dump method
objFile = open("AppData.dat", "ab")
pickle.dump(lstCustomer, objFile)
objFile.close()

# And, we read the data back with the pickle.load method
objFile = open("AppData.dat", "rb")
objFileData = pickle.load(objFile) #load() only loads one row of data.
objFile.close()

print(objFileData)
```

*Figure 2 Above we are presented with the ability to open a dat file that will store our eventual, appended binary output. To ensure that this data in indeed saved in binary, the **pickle.dump()** function is defined with the variables we want stored (**lstCustomer** and **objFile**). Since we then want to additionally read the stored data, we are also are implementing the **pickle.load()** function (_MOD7PYTHONPROGRAMMINGNOTES.DOCX 2022, PG 13)*

4. However, to first fully understand the above functions, I wanted to further explore Pickling in more depth. I visited AfterNerd (Karim, 2022). This website not only defined pickling in python, but also provided numerous examples in a tutorial format. A few concepts that were explored are as follows (and expanded upon with additional resources):
   - **a** Any object in Python can pickled for saving into memory, and pickling is the process of serializing or deserializing these objects. In more detailed terms, this is Python converting objects into character streams that contain all the info necessary to reconstruct the objects.
   - **b** Pickle is often opted for a variety of reasons: text files take more disk space and are not as scalable for schema purposes.
   - **c** Pickle is one of several serialization protocols available, with others being json or protocol buffers.
   - **d** Not everything can be pickled, but several topics we have explored so far in this class are compatible, such as: integers, floating-point numbers, strings, tuples, lists, dictionaries, functions, and classes (Python Software Foundation, 2022).

5. With the above in mind, I completed Lab 7-1. This will serve as a template for our eventual example of pickling in our Rabbit Management script!

```
import pickle  # This imports code from another code file!

# Data ---------------------------------------------- #
strFileName = 'AppData2.dat'
lstCustomer = []

# Processing ---------------------------------------- #
def save_data_to_file(file_name, list_of_data):
    file = open(file_name, "ab")
    pickle.dump(list_of_data, file)
    file.close()

def read_data_from_file(file_name):
    file = open(file_name, "rb")
    list_of_data = pickle.load(file)  # load() only loads one row of data.
    file.close()
    return list_of_data

# Presentation -------------------------------------- #
# TODO: Get ID and NAME From user, then store it in a list object
intId = int(input("Enter an Id: "))
strName = str(input("Enter a Name: "))
lstCustomer = [intId, strName]
print(lstCustomer)

# TODO: store the list object into a binary file
save_data_to_file(strFileName, lstCustomer)

# TODO: Read the data from the file into a new list object and display the contents
print(read_data_from_file(strFileName))
```

Figure 3 Note the incorporation of the Presentation section of our script to grab our user's input for appending to our file.

6. However, I next want to explore how I can add structured error handling into my script. The idea behind this concept is that we can trap errors in our program using a try-except block of code to assist with how our program handles errors versus letting Python choose its default action (_MOD7PYTHONPROGRAMMINGNOTES.DOCX 2022, PG 15). The need for this became apparent as I tested my developing Rabbit Management script, which I will detail further in our next few steps!

7. To get to our need for error handling and the trigger for exploring this concept more, I first continued forward with my vision for my Rabbit Management script, which is as follows: *a script that will take a user inputted ID and name of a rabbit, store it into memory via pickling, and then upon exiting our script, display back all our stored bunnies*. To streamline this, it made sense to

first tackle the functions shown in Figure 3. To do this, I edited my functions to the below:

```
import pickle

# Data ---------------------------------------- #
strFileName = 'RabbitManagement.dat'

# Processing & Pickling Example ---------------------------------- #
def save_data_to_file(file_name, list_of_data):
    file = open(file_name, "ab")
    for data in list_of_data:
        pickle.dump(data, file)  # Indication that data will be saved in binary via pickling
    file.close()

def read_data_from_file(file_name):
    file = open(file_name, "rb")
    list_of_data = pickle.load(file)  # load() only loads one row of data.
    file.close()
    return list_of_data
```

Figure 4 Note our edited two functions from Figure 3.

8. I then moved on to creating our presentation to the user that incorporated a loop for continual entry of bunnies or an option to exit our script.

```
# Presentation to User ---------------------------------- #
# Get Rabbit ID and Name from User
while(True):
    intId = int(input("Enter the Rabbit's ID #: "))
    strName = str(input("Enter Rabbit's Name: "))
    bunnyEntry = [intId, strName]
    print("...")
    save_data_to_file(strFileName, [bunnyEntry])
    print("Rabbit Saved!")
    print("****************************************")
    print("Hit any key to add more rabbits or type 'Exit' to review all")

    # If Option Exit is Selected
    strUserInput = input()
    if strUserInput == "Exit":
        break

print("****************************************")
print("All Current Buns:")

# Store the list object into a binary file
save_data_to_file(strFileName, bunnyEntry)

# Read the data from the file into a new list object and display the contents
print(read_data_from_file(strFileName))
```

Figure 5 Our user presentation that will collect the inputs of ID and name per rabbit via a while loop. These are the inputs that will eventually be pickled into storage via the functions we defined in figure 4.

9. Upon running the script created thus far, things seemed to be in working order, until the exit option was selected!

```
Enter the Rabbit's ID #: 8
Enter Rabbit's Name: Bun Bun
...
Rabbit Saved!
****************************************
Hit any key to add more rabbits or type 'Exit' to review all
Exit
****************************************
All Current Buns:
[1, 'Kona']
```

*Figure 6 Our script is not displaying all the rabbits stored in our file, just the first line. An expected error based off the note on **list_of_data = pickle.load(file)** from Figure 4. We are only loading one line at a time.*

10. This is an indication that we need to go back and edit our functions, and an additional opportunity to begin error handling presented itself. Our issue above is that we are loading data into a list and then attempting to read from that point. What we need to do is establish a continuous loop that will read all our appended data before closing out and returning all our entries at once. However, this can be tricky if starting this script fresh with no text in your file, because then you may receive an EOFError (Pansuriya, 2022). To achieve a solution to this, I edited my script to the following:

```
Processing & Pickling Example ------------------------------------ #
def save_data_to_file(file_name, list_of_data):
    file = open(file_name, "ab")
    for data in list_of_data:
        pickle.dump(data, file)  # Indication that data will be saved in binary via pickling
    file.close()

def read_data_from_file(file_name):
    file = open(file_name, "rb")
    list_of_data = []
    while(True):
        try:
            data = pickle.load(file)  # Indication that loaded data will be in binary
            list_of_data.append(data)
        except EOFError:
            break
    file.close()
    return list_of_data
```

*Figure 7 My edited **read_data_from_file** function with additional error handling for potential EOFErrors upon initial start of this loop. Assistance with formatting was provided with guidance from this week's module notes (_MOD7PYTHONPROGRAMMINGNOTES.DOCX 2022, PG 15).*

11. From here, I wanted to tackle the presentation of my stored rabbits since they currently were displayed back in a less than friendly manner:

```
Hit any key to add more rabbits or type 'Exit' to review all
Exit
****************************************
All Current Buns:
[[1, 'Kona'], [2, 'Blackberry'], [3, 'Cashew'], [4, 'Luna'], [5, 'Sunflower'],
```

Figure 8 Our current output of rabbits.

12. To organize the presentation of these rabbits, I created a new function to accompany my existing two.

```
def print_bunnies(list_of_data):
    for data in list_of_data:
        print(str(data[0]) + ": " + data[1])
```

Figure 9 A new function for better organizing the presentation of all our user inputted data.

13. To accompany the above function, I then edited the finale of my script with an accompanying print statement upon exiting the script.

```
    # If Option Exit is Selected
    strUserInput = input()
    if strUserInput == "Exit":
        break

print("****************************************")
print("All Current Buns:")
print_bunnies(read_data_from_file(strFileName))
```

Figure 10 Our new print statement for better review of our stored data.

14. This presents our rabbits in a much more organized manner!

```
...
Rabbit Saved!
****************************************
Hit any key to add more rabbits or type 'Exit' to review all
Exit
****************************************
All Current Buns:
1: Kona
2: Blackberry
```

Figure 11 Our resulting list of bunnies from our above print statement in Figure 12.

15. However, I still want to incorporate one additional error handling example. Ideally because I want my user to only be permitted to input an integer value for any given Rabbit ID.

```
# Presentation ---------------------------------- #
while(True):
    try:
        intId = int(input("Enter the Rabbit's ID #: "))
    except Exception as e:
        print("Invalid ID # input, please try again! Error: " + str(type(e)))
        continue
    strName = str(input("Enter Rabbit's Name: "))
```

Figure 12 Above is a demonstration of Exception Handling that presents our user with a custom message if they try to input a value that is not an integer for their Rabbit ID

16. We now have our script complete!

```
Enter the Rabbit's ID #: 3
Enter Rabbit's Name: Cashew
...
Rabbit Saved!
****************************************
Hit any key to add more rabbits or type 'Exit' to review all
Go
Enter the Rabbit's ID #: 4 Luna
Invalid ID # input, please try again! Error: <class 'ValueError'>
Enter the Rabbit's ID #: 4
Enter Rabbit's Name: Luna
...
Rabbit Saved!
****************************************
Hit any key to add more rabbits or type 'Exit' to review all
Exit
****************************************
All Current Buns:
1: Kona
2: Blackberry
3: Cashew
4: Luna
```

Figure 9 Completed Assignment 07. As shown in PyCharm.

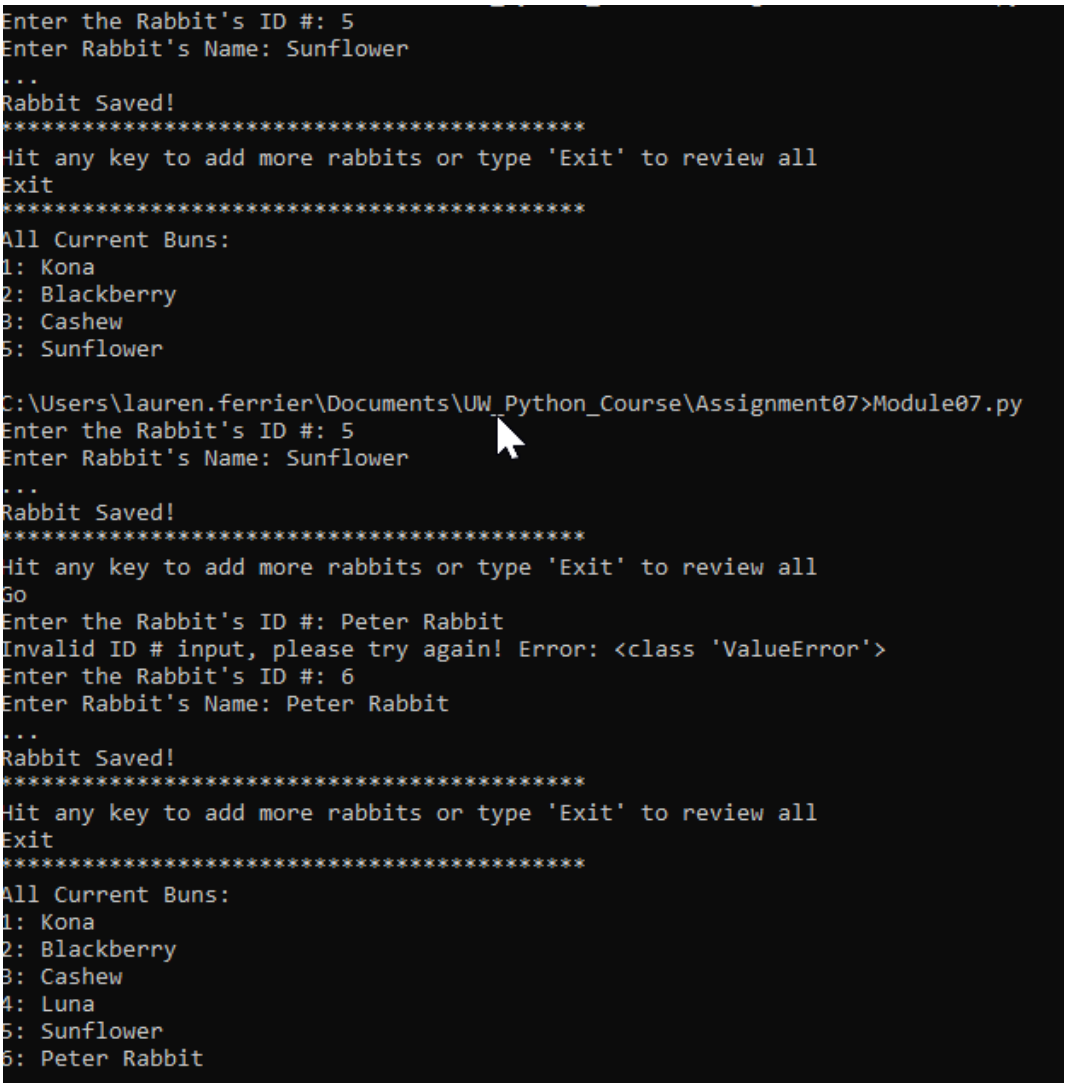

Figure 10 Completed Assignment 07, as shown in a command window.

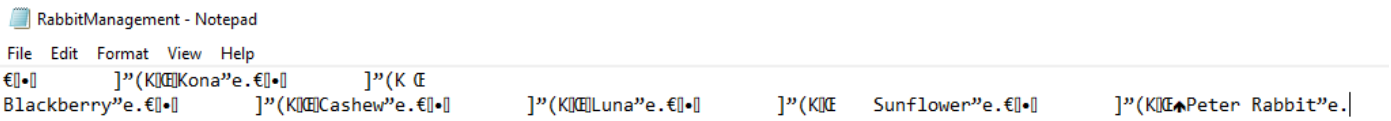

Figure 11 Resulting RabbitMamagement.dat from Assignment 07's executed script.

Summary

Using our provided course materials, I created a script that permits a user to store an appending list of rabbits in binary while additionally introducing exception handling. While previous week’s explored concepts used in this script such as loops and lists, Module 7 took this further by incorporating different methods of storing data and engaging with our users when it comes to interacting with our script.