Erin Leggett
November 30, 2021
IT FDN 110A
Module 07 - Assignment 07

ADVANCED DATA TYPES AND ERROR HANDLING

OBJECTIVE

Describe the steps taken to create the Python script attached to this assignment.

INTRODUCTION

Though we were briefly introduced to the subject in previous modules, this week we covered error handling in much greater detail - from understanding Python's built-in error handling functionality to printing and even defining and calling our own error messages. In addition to error handling, we also learned how to save and read data to and from binary files - a process referred to as pickling (and, respectively, unpicking) in the Python language. Finally, after performing some additional research we were asked to demonstrate these features in a tutorial-type script of our own choosing.

DEMO 1: PICKLING AND UNPICKLING

Our first task was to pickle and unpickle any data of our choosing in a basic demo script. After attending tonight's lecture, I greatly simplified my script to only handle two types of data: one string and one integer, comprising a "contact list" file very similar to our previous inventory file. I had to start, though, by importing the pickle module:

```
import pickle
```

From there, the script uses two basic input statements to collect a contact name and contact number from the user. To keep things easy and ensure I was working with disparate data types, I manually defined these values as str and int, respectively, when collecting them:

```
contact_name = str(input("Enter a name to add to contact list: "))
contact_number = int(input("Enter a phone number for your contact
(numbers only, please!): "))
```

Once these data points have been collected, they are organized into a list file, then a dictionary object, and then saved back to the Assignment07.dat binary file (or *pickled)* using the pickle.dump() function:

```
myfile = open("Assignment07.dat", "wb")
pickle.dump(contact_dict, myfile)
myfile.close()
```

Furthermore, to ensure the data indeed wrote back to the binary file, one, and two was recorded as organized and intended, I had the script read the file back and display the dictionary data without any additional adjustment or formatting on my end. This process is referred to as unpicking, accomplished using the pickle.load():

```
myfile = open("Assignment07.dat", "rb")
mydata = pickle.load(myfile)
print(mydata)
myfile.close()
```

DEMO 2: STRUCTURED ERROR HANDLING

Next, we were tasked with demonstrating structured error handling. To do so, I used the integer and string data defined in the previous demonstration to try to construct a (purposefully invalid) data type via concatenation. In order to better illustrate the difference between custom and default error handling, I had my first try-except block print the text of the Python-generated error message as well as its type when the `new_list` data point I was trying to create inevitably failed to complete, but little else:

```
print(e, type(e), sep = "\n")
```

In contrast, I not only created a more robust try-except block using custom-generated error messages tailoring to a few particular error classes, but also included more thorough system information such as the docstring in the error report:

```
try:
    new_list = contact_name + contact_number
    print(new_list)
except TypeError as e:
    print("\n" + "Can't concatenate numeric and string data! Error
report: " + "\n")
    print(e, e.__doc__, e.__str__,type(e), sep = "\n")
except FileNotFoundError as e:
    print("\n" + "Assignment07.dat file must exist prior to running
this script! Error report: "+ "\n")
    print(e, e.__doc__, e.__str__, type(e), sep="\n")
except Exception as e:
    print("\n" + "An unspecified error occurred! Error report: " +
"\n")
    print(e, e.__doc__, e.__str__, type(e), sep="\n")
```

DEMO 3: CUSTOM EXCEPTION CLASSES

Finally, to weave these two concepts together I re-hashed the pickling demonstration with some added functionality: first, to allow for more expressive errors, I allowed the user to input whatever they liked as a name and contact number; and secondly, to ensure the user did not enter any incorrect data, I had the try-except blocks prevent any data from being picked or unpicked unless it met my basic criteria. Which is to say, in order for the user to get through to saving and reading their binary file, they would have to input a string for a name and a 10-digit phone number as integers - no exceptions. To accomplish this, I nested the pickling and unpicking functions within their own try-except block, preceded by a few others to restrict the user's inputs:

```
try:
    contact_name = input("Enter a name to add to contact list: ")
    if contact_name.isnumeric():
        raise AlphaError()
    elif len(contact_name)==0:
        raise EntryError()
except Exception as e:
    print("\n" + "Error report: " + "\n")
    print(e, e.__doc__, e.__str__, type(e), sep="\n")

try:
    contact_number = input("Enter a phone number for your contact
(numbers only, please!): ")
```

```
        if contact_number.isalpha():
            raise NumError()
        elif len(contact_number)!=10:
            raise PhError()
        else:
            contact_list = [contact_name, contact_number]
            contact_dict = [{"Name": contact_name, "Number":
    contact_number}]
            myfile = open("Assignment07.dat", "wb")
            pickle.dump(contact_dict, myfile)
            myfile.close()
            myfile = open("Assignment07.dat", "rb")
            mydata = pickle.load(myfile)
            print(mydata)
            myfile.close()
    except Exception as e:
        print("\n" + "Error report: " + "\n")
        print(e, e.__doc__, e.__str__, type(e), sep="\n")
```

As a means of preventing the user from inputting undesirable data (i.e., a string in the place of a numeric field, or vice versa), I even called in a few custom exception *classes* to be employed when specific conditions were met. For example, if the user inputs an integer into the contact_name field (or: `if contact_name.isnumeric()`), a custom-defined `AlphaError` message will be raised prompting the user to input only alphanumeric characters into the name field.

CODE VALIDATION

To make the script run more like a “tutorial”, I ran back through it to add print and input statements to space out functionality and pause the script between demonstrations. Depending on user input, the script can throw a number of custom exceptions; however, for the sake of validation I have included screenshots of successful demonstration completion in both PyCharm and Terminal, as well as a full picture of the script flowing in the IDE:

```
Enter a name to add to contact list: Bob Smith
Enter a phone number for your contact (numbers only, please!): 123456

Here is our data as a row in a list:
['Bob Smith', 123456]

And here's that list converted into a dictionary:
[{'Name': 'Bob Smith', 'Number': 123456}]

Saving dictionary data binary file... (= PICKLING)
Assignment07.dat successfully saved!

Reading dictionary data from binary file... (= UNPICKLING)
[{'Name': 'Bob Smith', 'Number': 123456}]

Pickling tutorial complete!
```

```
Press Enter to continue to error handling demonstration.

We'll use the inputs from our pickling demonstration to produce a Python error.
Here's what happens when we try to concatenate our data instead of placing it into a list object:

can only concatenate str (not "int") to str
<class 'TypeError'>

Now, let's leverage exception classes and built-in functions to add more detail. Press Enter to try again.

Can't concatenate numeric and string data! Error report:

can only concatenate str (not "int") to str
Inappropriate argument type.
<method-wrapper '__str__' of TypeError object at 0x10d310680>
<class 'TypeError'>

Error handling tutorial complete!
```

```
Press Enter to re-run pickling script with custom error handling.

Enter a name to add to contact list: Bob Smith
Enter a phone number for your contact (numbers only, please!): 1234567890

Valid entry! Name and number added to list data.

Assignment07.dat successfully overwritten!

Assignment07.dat successfully read:
[{'Name': 'Bob Smith', 'Number': '1234567890'}]

Assignment07.dat closed. Thank you for using my tutorial. Goodbye!
```

```
Press Enter to re-run pickling script with custom error handling.

Enter a name to add to contact list: Jane Doe
Enter a phone number for your contact (numbers only, please!): 0987654321

Valid entry! Name and number added to list data.

Assignment07.dat successfully overwritten!

Assignment07.dat successfully read:
[{'Name': 'Jane Doe', 'Number': '0987654321'}]

Assignment07.dat closed. Thank you for using my tutorial. Goodbye!
Pete-Wisdom:Assignment07 erinleggett$
```

LESSONS LEARNED

For my first attempt at this assignment, I tried to take last week's assignment and mold it into a demonstration of pickling/unpickling and structured error handling; however, that proved to be a bit too complicated, especially when it came to comparing entries against the binary list to find a match for removal. However, further clarification from tonight's lecture made me realize we were only meant to *demonstrate* these features, not necessarily create a fully-functional, interactive script. To minimize confusion with half-working functions and a multitude of already-defined variables, I started over, but was able to debug with much greater speed having unsuccessfully fought to force these features into last week's assignment already.

RESOURCES

Han, Na Rae, *Pickling,* accessed 29 November 2021,<https://sites.pitt.edu/~naraehan/python3/pickling.html>.

Kalia, Manu, 2019, *Don't Fear the Pickle: Using pickle.dump and pickle.load*, accessed 30 November 2021, <https://betterprogramming.pub/dont-fear-the-pickle-using-pickle-dump-and-pickle-load-5212f23dbbce>.

Python, pickle — Python object serialization*,* accessed 29 November 2021,<https://docs.python.org/3/library/pickle.html>.

GITHUB LINK

https://emleggett.github.io/IntroToProg-Python-Mod07