Tim Shore
November 30, 2020
Foundations of Programming: Python
Assignment 07
ITFnd100-Mod07/index.md at main · timothyshore/ITFnd100-Mod07 (github.com)

Pickling and Structured Error Handling with Python

Introduction

As our programs in Python grow in depth and length, there are a couple of additional features available that will be most helpful in keeping the programs on track. Large amounts of data can be saved and imported to and from a Python program with "pickled" files saved as binary data files.

What is a Pickle in Python?

So far in this class, we've saved and wrote our data from the Python programs to a text (.txt) file. By using the pickle module, we can save and upload data from binary files (.dat). "The pickle module implements binary protocols for serializing and de-serializing a Python object structure." (*python.org* https://www.docs.python.org/3/library/pickle.html 2020)(External Site).

Dumping and Loading a Pickle

To serialize (or write) a binary file, we use the **dump()** function after the pickle module. In the parentheses you would indicate the object then file separated by a period.

pickle.dump(object, file_name).

You want the file name to be a binary file with the extension .dat. When the programmer opens the file, the letter b is added to the a(append) or w(overwrite) command.

file_name = open('File.dat', 'ab')

Then you close the file when you are done with the dump.

To de-serialize (or read) a binary file, the programmer uses the **load()** function after the pickle module. First you identify , you only need to define the file that is being read within the parentheses. And again, you add the letter b (for binary) after the r(read) command.

file_name = open('File.dat', 'rb')
pickle.load(file_name)

What Can Be Pickled

Programmers can pickle all kinds of objects including numbers, strings, tuples, lists and dictionaries. One thing to keep in mind: pickle modules are not secure. You only want to import a pickle that you know comes from a source you trust.

Structured Error Handling in Python

So far with the scripting of code in Python, we've discovered an error when running the program when it cannot execute the code. By scripting an **exception**, Python is able to respond to the errors when they occur. "An exception is an error that happens during execution of a program. When that error occurs, Python generates an exception that can be handled, which avoids your program to crash." (*pythonforbeginners*, https://www.pythonforbeginners.com/error-handling/exception-handling-in-python 2020)(External site).

Handling an Exception

By using a **try-except** block of code, we can trap the errors in the program and customize how the program handles the errors. After the try command, we specify what the user might try to accomplish that would be considered an error. The code is indented below the **try:** command so it is read as part of the command. Then we script an **except:** command telling the program what it should do when the error is encountered. The code for that is indented below the except: command. However, it is good practice to specify the exception type on the except line, otherwise the program will catch all exceptions that might be triggered by the error. Try statements can have multiple except statements to capture multiple exceptions.

Raising an Exception

You can call a specific exception with the **raise** statement. "The raise statement allows the programmer to force a specified exception to occur." (*python.org*, https://www.docs.python.org/3/tutorial/errors.html 2020)(External site). The raised exception can be from the exception classes already within the Python program, or it can even be a custom created class by the programmer. The raise statement is included within and at the end of the try statement (which can also have several "tries" by using the if-else statement – however each of the if-else statements should have the raise statement of which exception to use). If a programmer wants to know which of the built-in exception classes should be used, the python Program will tell you when it hits the error on the "run" screen.

Scripting the Assignment

Both Examples in One File

The best way to see how these two new concepts work, is to show it in this week's assignment. I've maintained using the Separation of Concerns (SoC) as Data, Processing, and Presentation. I used the comment hashtag above the code that was for either the Pickle code or the Structured Error Handling.

Pickle

For my pickle module, I had the user enter three separate strings of information into a list and then saved it or "dumped" it into a binary file. The program asked for a task, then the priority, and since it was a task for the care of a dog, I asked if it required a leash or no leash. I then read or "loaded" the list back into the program and printed it out for the user to see.

Structured Error Handling

For the structured error handling example, I used a numeric error code. Since the error already had a class in the

Python program, I not only printed out my own error message, but also called up the existing error message from Python and had that printed out as well. Below are screen captures of the program running in PyCharm and the Command Line Prompt with their corresponding binary files as seen in a .txt file.

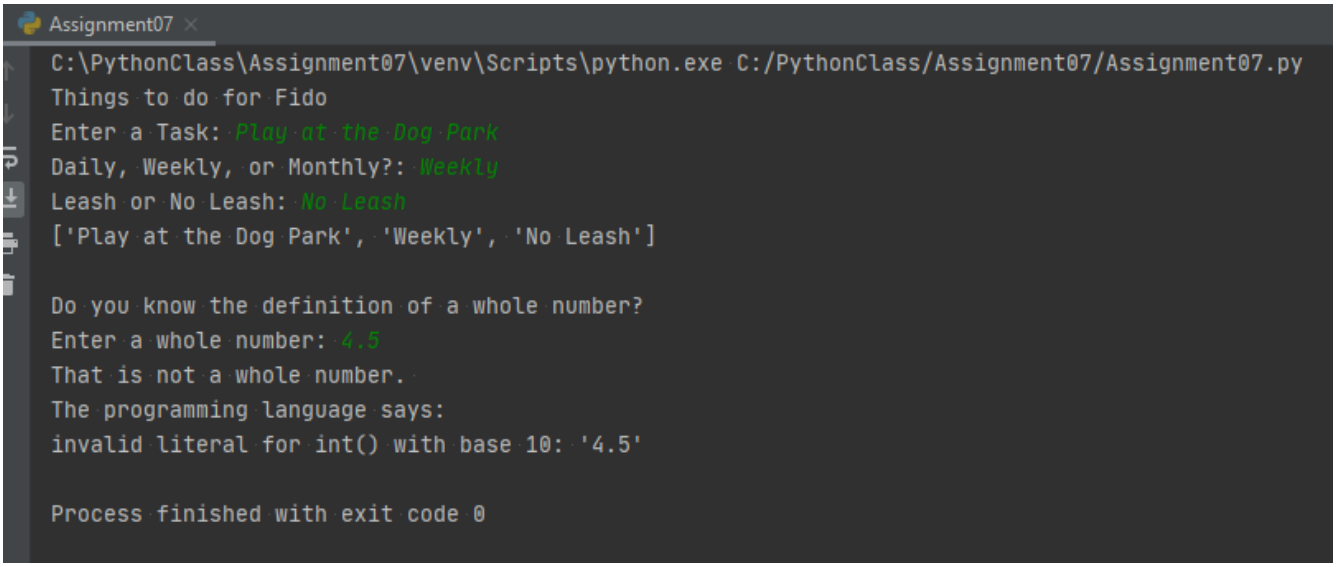

Figure 7.1: Running the code in PyCharm

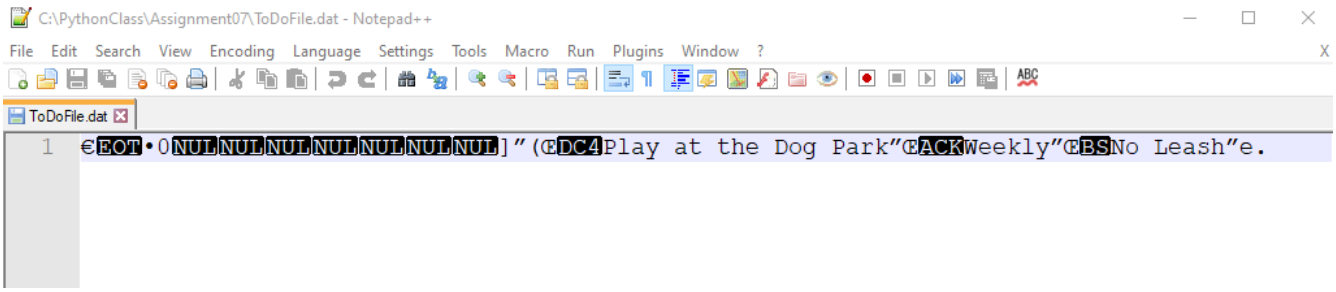

Figure 7.2: Binary file saved from PyCharm in a Notepad++ file

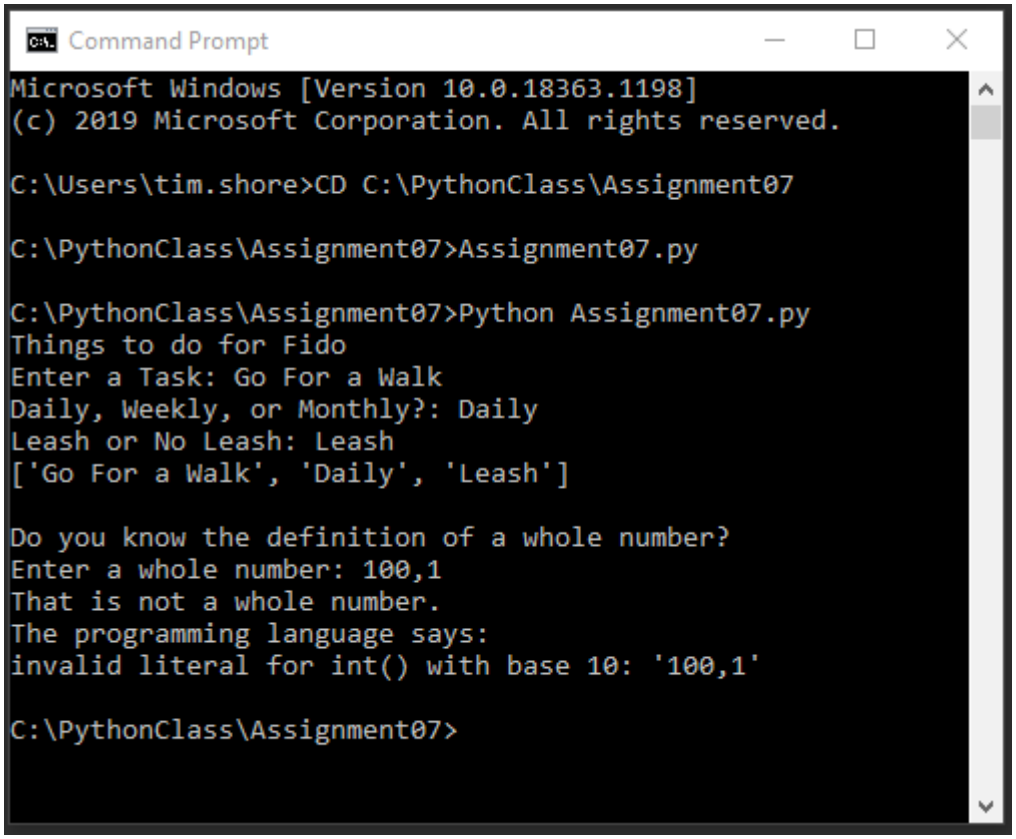

Figure 7.3: Running the Program in a Command Line Prompt Window

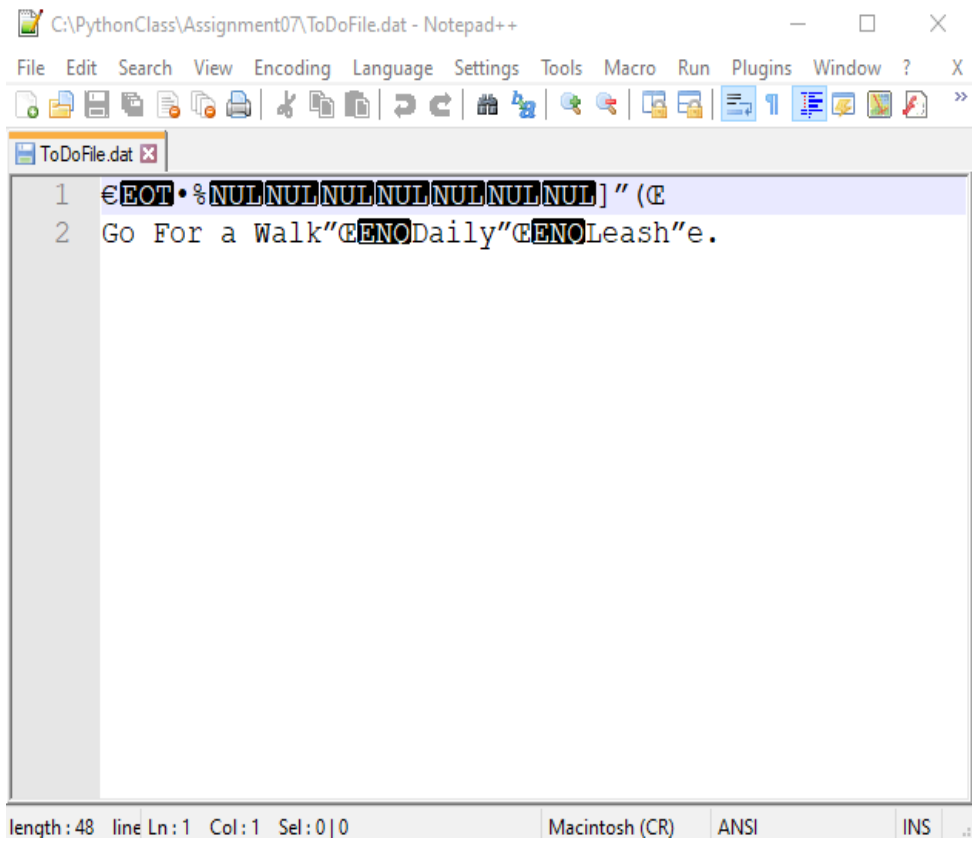

Figure 7.4: Binary File saved from Command Line Prompt in a Notepad++ File

Summary

The program that I created to illustrate the Pickle and Structured Error Handling were rather small. But when considering scripting much larger files, these tools will be invaluable when imagining the algorithms and then coding the files for use. Pickling for storing and retrieving data and Structured Error Handling to anticipate errors that may occur when a user interacts with the program.