How Pickling and Structured Error Handling Work

**Introduction and Script**

My explanations for each are inside the code itself, so here I’ll go over what I’ve learned about each. Otherwise I’ll be repeating what is already in my code.

**Code Without Header**

# Variables

house1 = {}

# Processing

def picklingdemo():

# Dictionary

import pickle

house1 = {'key':'Lakewood','location':'north','stories':'two'}

print(house1)

pickle\_out = open("dict.pickle","wb")

pickle.dump(house1,pickle\_out)

pickle\_out.close()

def provepickle():

import pickle

pickle\_in = open("dict.pickle","rb")

house1 = pickle.load(pickle\_in)

print(house1)

print(house1['location'])

def divide(x,y):

try:

equals = x / y

except ZeroDivisionError:

print("Cannot divide by zero!")

else:

print("Equation equals",equals)

finally:

print("This text will show up even when errors occur")

# Presentation

def introduction():

print("======================================================\n"

"\tHello User, this script attempts to explain how\n"

"\tPickling and Structured Error Handling work using\n"

"\tPython. The first part will be how pickling works\n"

"\tand the second will be an example of structured\n"

"\terror handing. There will be stop points that\n"

"\trequire you to hit the enter key to continue.\n"

"\t\t\*\*\* NOTICE \*\*\*\n"

"\tThis is done by a very amature programmer, the\n"

"\tfollowing information may not be fully correct.\n"

"======================================================")

def proceed():

input("\n"

"----- Please Press 'Enter' to Continue -----\n")

def picklingex():

print("\tPicking data is taking object data and turning\n"

"\tit into a byte stream. A byte stream is an ordered\n"

"\tsequence of bytes, which is displayed or saved\n"

"\tin a binary file. Two purposes for doing this are\n"

"\tto store information in a file/database or send\n"

"\tit over a network.\n\n"

"\tThe following are able to be pickled:\n"

"---------------------------------------------\n"

"\tNone, true, and false\n"

"\tIntegers, long integers, floating point numbers, complex numbers\n"

"\tNormal and unicode strings\n"

"\tTuples, lists, sets, and dictionaries containing only picklable objects\n"

"\tFunctions defined at the top level of a module\n"

"\tClasses that are defined at the top level of a module\n"

"---------------------------------------------\n"

"\t\t\*\*\* !!!ATTENTION!!! \*\*\*\n"

"\tThere is no effective way to verify pickled data being unpickled.\n"

"\tONLY unpickle data from absolutly trusted sources. Raw pickled data\n"

"\tmay be malicious, unpickle at your own risk. Never use it with\n"

"\tuntrusted sources.")

def explinationpickle():

print("\n\tThe above imports picking, takes a dictionary to save to a pickled file,\n"

"\tthen unpickles the file along with extract a specific piece of info\n"

"\tto prove it works. This ends the explination for pickling.")

def errorhandlingintro():

print("\tStructured error handling is writing the code in such a way\n"

"\tthat if an error occurs there is an built in reaction to it.\n"

"\tA way to do this would be using 'try' along with 'except'\n"

"\tstatements that have the error type in them. An additional step\n"

"\twould be use 'else' statements if the error is outside of what\n"

"\twas already planned.\n\n"

"\tAn example of this would look like =\n"

"\t\ttry:\n"

"\t\texcept:\n"

"\t\telse:\n"

"\t\tfinally:")

def errorhandlingex():

print("\n\tThe above makes use of try, except, else, and finally to\n"

"\tmanage a divide by zero error. The math equation tried is 2/0\n"

"\twhich will cause an error. When the error occurs it\n"

"\tprints out something that is easier for someone who\n"

"\tdoesn't read code to understand. If the error is something\n"

"\telse the finally part will always show. The finally\n"

"\tstatement will allow the code to keep running instead\n"

"\tof stop when an error occurs.")

def conclution():

print("\tThis concludes how pickling and structured error handling\n"

"\twork along with an example for each. Please read the code\n"

"\tto get more detail as to whats going on in the background.\n"

"\n\tThank you and have a great day! The next 'Enter' will close\n"

"\tthe script.")

# Body of Script

introduction()

proceed()

picklingex()

proceed()

picklingdemo()

provepickle()

explinationpickle()

proceed()

errorhandlingintro()

proceed()

divide(2,0)

errorhandlingex()

proceed()

conclution()

proceed()

**Part 1**

The first part would be pickling, which was a bit more straightforward in practice that I was anticipating. It’s the explanation that I’ll probably understand better in time. As of currently I don’t have much of a solid perception of bytes or the concept of binary. What I learned is this is a great way to speed up the process of transferring data somewhere due to it being less complicated when pickled. Supposedly it can speed up that process by one hundred times or more, so says sources I had to find. At the amature level I am at right now I’ll probably use it to store and retrieve data quicker, or to simply practice with.

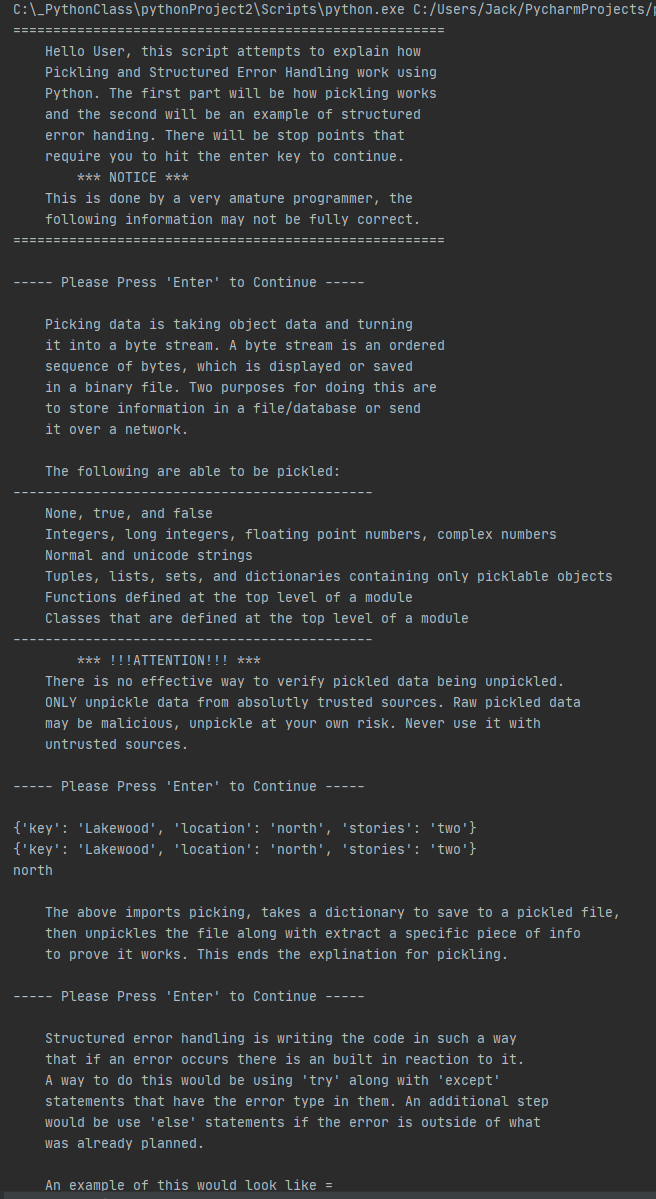
**Part 2**

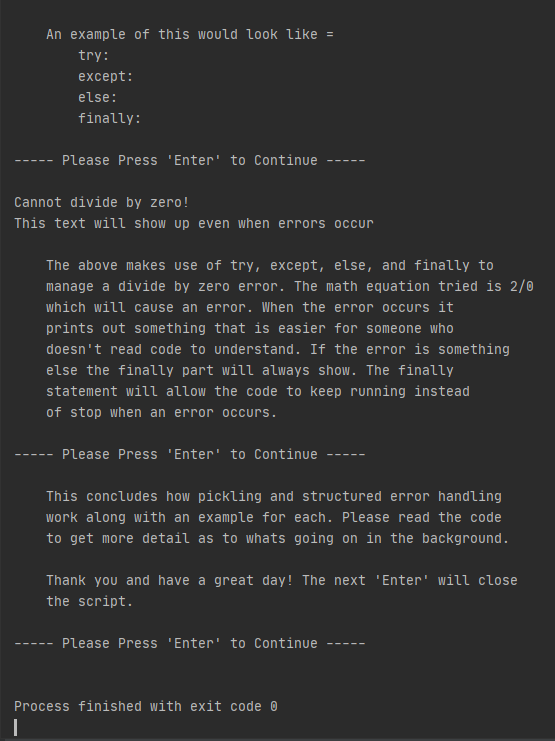
This part is structured error handling, and I feel strongly that this is an area of technique that I’m simply going to have to keep on coding to learn better. All the errors and exceptions aren’t known to me so I can’t know all of what I would have to create ‘except’ statements for. It is really nice to have these though, as having my own text instead of big intimidating red text just looks better. This kind of code could go anywhere I’m expecting a user to add or manipulate data, then predict what might happen.

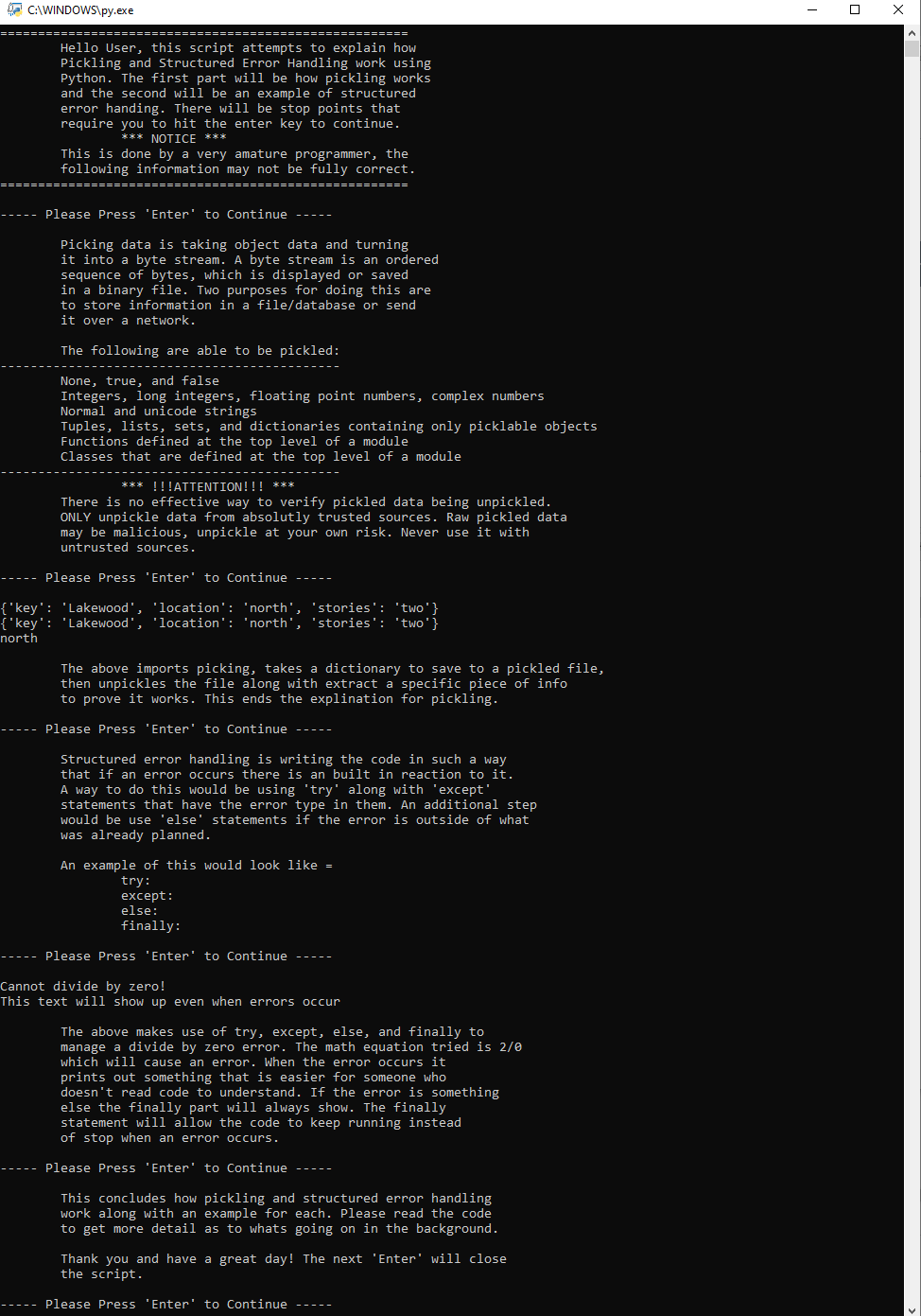
**Conclusion**

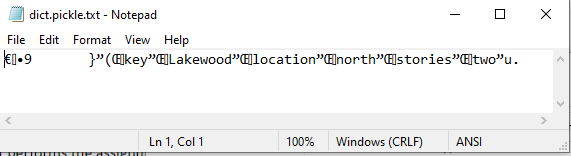
There are definitions and examples in the script itself, this was mainly what I learned during the process. It is definitely seen how these two things may be used in code, and better understood when I continue coding. At this point I’ve got an understanding but not a full grasp.

**Proof**

****

****

****

****