Michelle Diaz
February 25, 2023
IT FDN 110 A Wi 23: Foundations of Programming: Python
Assignment07
https://github.com/mdiaz1122/IntroToProg-Python-Mod07

Assignment 07: Pickling and Error Handling

Introduction

This document explains what pickling and error handling are in Python and provides examples of how they are utilized in programs.

Pickling

What is it?

At a very basic level, pickling can be described as preserving information for storage or transport in a binary format. This obscures the information to the naked eye but does not secure the information via encryption. Pickling can only be done to certain data types including booleans, integers, floating-point numbers, complex numbers, strings, tuples, lists, and even dictionaries if the dictionary objects are pickleable (Source: https://docs.python.org/3/library/pickle.html#what-can-be-pickled-and-unpickled). There are ways to write data to a text file from a Python program, but pickling allows you to save several objects serially for later retrieval and use in another Python script. Pickling also preserves the object's data type. So if you pickle a dictionary object, you will unpickle a dictionary object as well so this should be kept in mind when unpickling to variables you are using later.

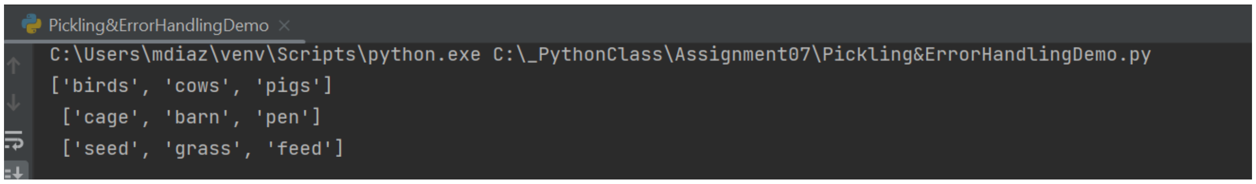
How do you use it?

In general, the two main commands for pickling (or storing) and unpickling (or loading) are pickle.dump(object) and pickle.load(object). There is more advanced functionality in the pickle module that can be viewed on the Python website, which does an excellent job at explaining the details and nuances of the pickling module (Source: https://docs.python.org/3/library/pickle.html#). For now, we will focus on the basic use of the two main pickle functions.

Below is an example of how pickling and unpickling work. Note that to use the pickle module, it must be imported at the beginning of the program. In this example, I define three lists and store them in three different variables called 'animals', 'dwelling', and 'food'. I then write these three list objects to a binary file and close the file when I am done. This is when the pickling has been completed. To demonstrate how unpickling works, I then load the list objects from the binary file to three new variables called 'animals2', 'dwelling2', and 'food2' and print the new variables. Printing the contents of the three new variables demonstrates that the values have been preserved from the pickling operation carried out in the first part of the script. Note how the variables are populated/loaded/unpickled in the same order that they were pickled in.

```python
import pickle

# Define Three List Objects
animals = ["birds", "cows", "pigs"]  # A list of farm animals
dwelling = ["cage", "barn", "pen"]  # A list of farm animal dwellings
food = ["seed", "grass", "feed"]  # A list of farm animal food

# Store List Objects in Binary File
fileout = open('farm_info.dat', "wb")
pickle.dump(animals, fileout)
pickle.dump(dwelling, fileout)
pickle.dump(food, fileout)
fileout.close()

# Load List Objects from Binary File
filein = open('farm_info.dat', 'rb')
animals2 = pickle.load(filein)
dwelling2 = pickle.load(filein)
food2 = pickle.load(filein)

# Print Lists from New Variables
print(str(animals2)+"\n", str(dwelling2)+"\n", str(food2)+"\n")
```

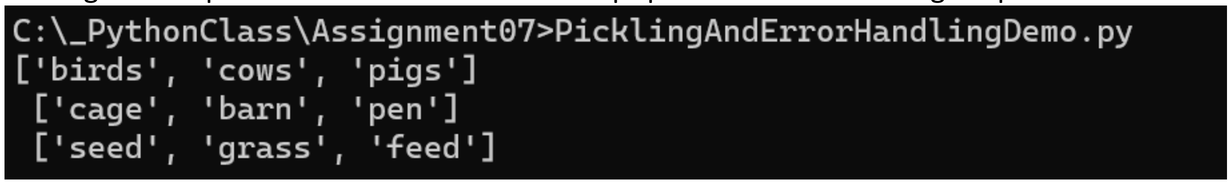
Running this script in PyCharm produces the following output:



```
C:\Users\mdiaz\venv\Scripts\python.exe C:\_PythonClass\Assignment07\Pickling&ErrorHandlingDemo.py
['birds', 'cows', 'pigs']
 ['cage', 'barn', 'pen']
 ['seed', 'grass', 'feed']
```

Figure 1: Pickling and Unpickling Script Output in PyCharm

Running this script in Windows Command Prompt produces the following output:



```
C:\_PythonClass\Assignment07>PicklingAndErrorHandlingDemo.py
['birds', 'cows', 'pigs']
 ['cage', 'barn', 'pen']
 ['seed', 'grass', 'feed']
```

Figure 2: Pickling and Unpickling Script Output in Command Prompt

From this example we can see how pickling is beneficial in storing and restoring variable values for later use in Python code. This can be done with any pickleable object and can be quite useful in some applications. However, it should be remembered that pickling is not a secure way of storing information and that binary files from external sources should be unpickled only if the source is trustworthy to prevent getting viruses.

Error Handling

What is it?

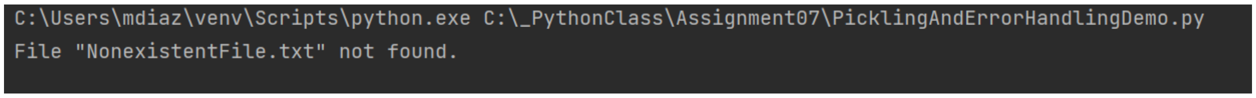
In Python, generally when errors occur the script or program is halted and an error message is displayed to the user. Sometimes these errors are hard to understand or could be avoided by coming up with a way to handle certain errors automatically. For example, if a script attempts to open a file with a filename that does not exist in the directory, a FileNotFoundError will be displayed and the program will exit. In these cases, you may just want the program to continue with the file creation since it does not already exist. Or if the file name was a user input, then you may want to prompt the user for a different file name that exists within the directory. Exceptions can be made for several error types to prevent a program from crashing. One such way to do this is by using ‘try’, ‘except’, ‘else’, and ‘finally’ blocks. The Python documentation page on error handling does a good job of explaining these blocks further (https://docs.python.org/3/tutorial/errors.html#handling-exceptions).

How do you use it?

The simplest way to handle errors in a program is to use the try/except/else/finally blocks. Some examples of errors that can occur are ValueError, ZeroDivisionError, and FileNotFoundError. An example of the syntax of try/except blocks can be seen below. This script simply tries to open a file that does not yet exist in the directory and since the file is not found, a FileNotFoundError is raised. The except block handles this specific error by printing a message to the user notifying them that file could not be found.

```
# Try/Except Example
try:  # Tries to open the file with name "NonexistentFile.txt"
   file = open(file_name, "r")
   file.close()
except FileNotFoundError:  # Adds provision for when no file exists yet
   print("File " + '"' + str(file_name) + '"' + " not found.")  # Notifies the user that the file was not found
```

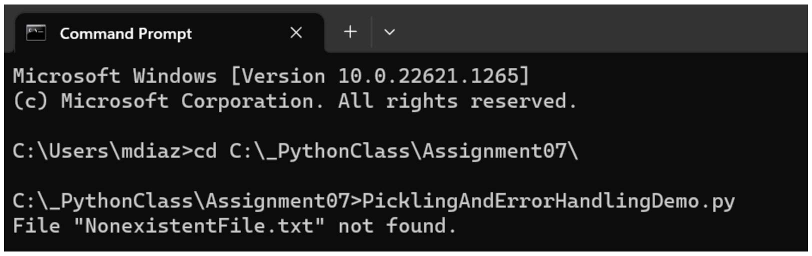
Note that the ‘except’ block can be used without a specific error specified before the colon. This will just apply whatever is in the ‘except’ block to any error that is raised and not just a specific type of error. Running this script in PyCharm results in the following output:



```
C:\Users\mdiaz\venv\Scripts\python.exe C:\_PythonClass\Assignment07\PicklingAndErrorHandlingDemo.py
File "NonexistentFile.txt" not found.
```

Figure 3: Error Handling for FileNotFoundError in PyCharm

Running this script in Windows Command Prompt produces the following output:



```
Microsoft Windows [Version 10.0.22621.1265]
(c) Microsoft Corporation. All rights reserved.

C:\Users\mdiaz>cd C:\_PythonClass\Assignment07\

C:\_PythonClass\Assignment07>PicklingAndErrorHandlingDemo.py
File "NonexistentFile.txt" not found.
```

Figure 4: Error Handling for FileNotFoundError in Command Prompt

In general, the exception handling blocks are executed with the following logic (Source: https://www.tutorialsteacher.com/python/exception-handling-in-python):

```
try:
   #statements in try block
except:
   #executed when error in try block
else:
   #executed if try block is error-free
finally:
   #executed irrespective of exception occurred or not
```

Conclusion

In conclusion, the pickling module and error handling features can be powerful tools in building an advanced Python program in which objects are preserved and re-used and errors are handled behind the scenes instead of crashing the program when they occur.