Haley Barsa
August 26, 2022
IT FDN 110 B Foundations of Programming: Python
Assignment07
https://github.com/hbarsa/IntroToProg-Python-Mod07

Demonstration of Pickling and Structured Error Handling

Introduction

This document presents the steps taken to write a script which demonstrates pickling and structured error handling. PyCharm is used to edit and test the script. IDLE is also used to test the script. Classes, objects, pickling, and error handling are discussed in the following sections.

Classes and Objects

To understand pickling, it is first helpful to understand the relationship between classes and objects. A class is like a blueprint for a house and the object is the house. Many houses can be created from the same blueprint, just as many objects can be created from the same class. An object is an instance of a class. Said in other words, an object is a copy of the class with actual values.

Pickling

Pickling is built into the Python standard library and is used for serializing and deserializing objects. This can be useful if you have a large dataset which takes substantial time to load into memory. If you first pickle the data and then load it, it will be much faster. Pickling also freezes the state of the object so that it remains an exact copy from when you pickled it, hence the name pickling. All data types can be pickled (ex: integers, strings, tuples). Serialization is initiated with the pickle.dump() or pickle.dumps() commands and deserialization is initiated with the pickle.load() and pickle.loads() commands. The serialized data can be stored in a file or a byte string. Include an “s” for byte string objects and leave off the “s” for file type objects. See Figure 1 below for example script pickling an object into serialized format.

```
my_pickled_object = pickle.dumps(my_object) #pickle the instance
```

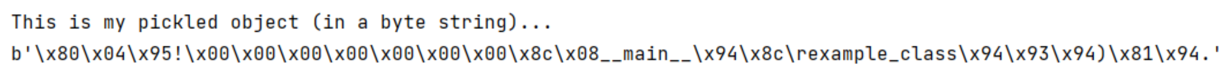
Figure 1: Pickle Data

See Figure 2 below for example script converting serialized format back into the original data type.

```
my_unpickled_object = pickle.loads(my_pickled_object) #unpickle back to Python object
```

Figure 2: Unpickle Data

The pickled data is illegible. See Figure 3 for an example of a pickled byte string.



```
This is my pickled object (in a byte string)...
b'\x80\x04\x95!\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\rexample_class\x94\x93\x94)\x81\x94.'
```

Figure 3: Pickled Byte String

Structured Error Handling

To test that the pickled object can not be changed, the input function is used to collect a new number and change the value of the number in the class. Gathering input for the user is a great opportunity to use structured error handling. The script requires a numeric character type to perform correctly. Therefore, error handling is added to guide the user. A custom exception class is created since in addition to being numeric, a different number is requested to demonstrate the pickling process. A custom exception class allows you to raise an error for a specific situation and output a custom message. See Figure 4 for an example of a custom exception class.

```
class NewNumberError(Exception):
    """  Error message for entering the same number  """
    def __str__(self):
        return 'You entered the same number. Please enter a different number.'
```

Figure 4: Custom Exception Class

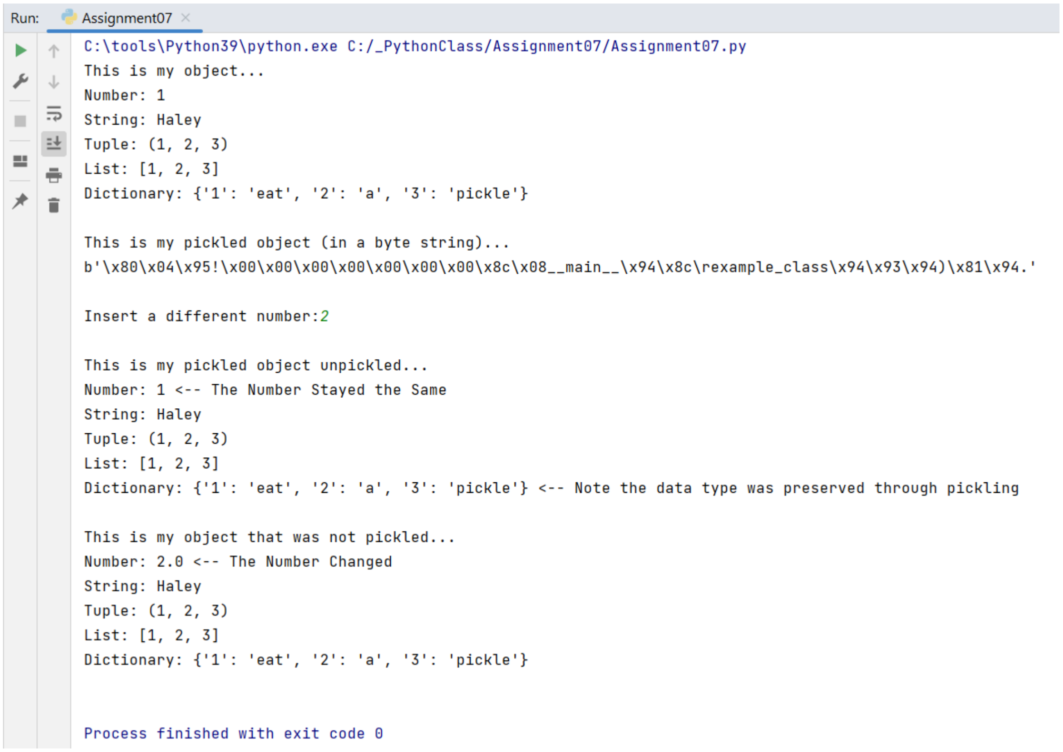
Inputting a non-numeric character will result in a ValueError since I have used the float() command to return a floating point number. Therefore, I added error handling to specifically catch a ValueError. See Figure 5 for raising the custom message and catching specific errors.

```
try:    #enter new data
    new_number = float(input("Insert a different number:"))
    print("")
    if float(new_number) == float(my_object.example_number):
        raise NewNumberError()
except ValueError as e:
    print("You entered non-numeric characters. Please enter a number.")
```

Figure 5: Catching Specific Errors

Pickle Demonstration

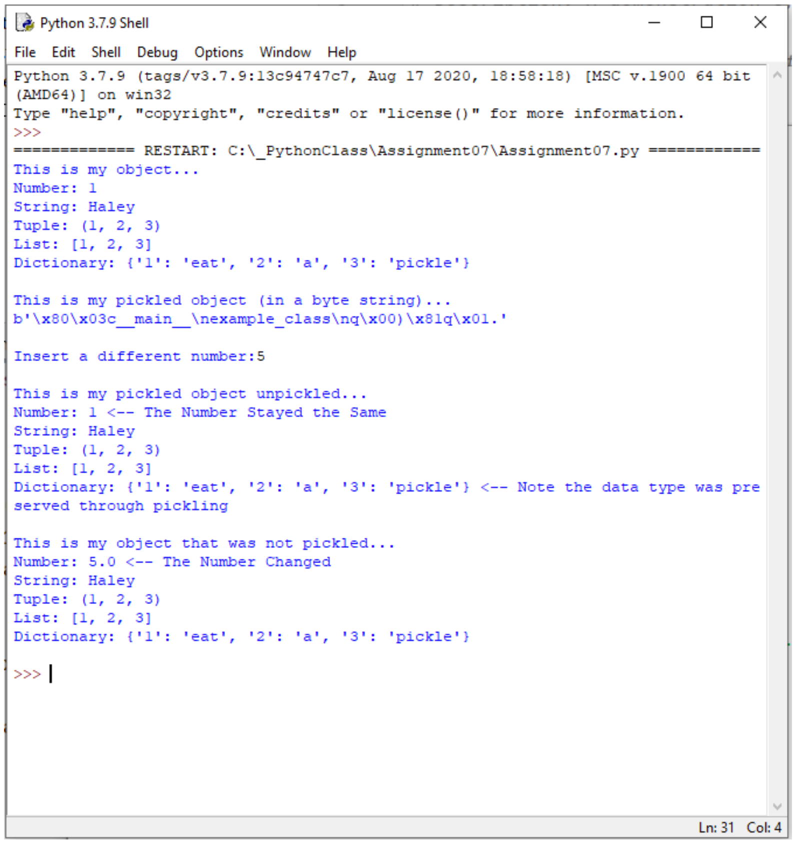
The code demonstrates how a pickled object cannot be changed but a non-pickled object can. See screenshot in Figure 6 of the program running.



```
C:\tools\Python39\python.exe C:/_PythonClass/Assignment07/Assignment07.py
This is my object...
Number: 1
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'}

This is my pickled object (in a byte string)...
b'\x80\x04\x95!\x00\x00\x00\x00\x00\x00\x00\x8c\x08__main__\x94\x8c\rexample_class\x94\x93\x94)\x81\x94.'

Insert a different number:2

This is my pickled object unpickled...
Number: 1 <-- The Number Stayed the Same
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'} <-- Note the data type was preserved through pickling

This is my object that was not pickled...
Number: 2.0 <-- The Number Changed
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'}


Process finished with exit code 0
```

Figure 6: Script Running From PyCharm

Figure 7 below shows the script working in a shell window.



```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============= RESTART: C:\_PythonClass\Assignment07\Assignment07.py ============
This is my object...
Number: 1
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'}

This is my pickled object (in a byte string)...
b'\x80\x03c__main__\nexample_class\nq\x00)\x81q\x01.'

Insert a different number:5

This is my pickled object unpickled...
Number: 1 <-- The Number Stayed the Same
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'} <-- Note the data type was pre
served through pickling

This is my object that was not pickled...
Number: 5.0 <-- The Number Changed
String: Haley
Tuple: (1, 2, 3)
List: [1, 2, 3]
Dictionary: {'1': 'eat', '2': 'a', '3': 'pickle'}

>>> |
```

Figure 7: Script Running From a Shell Window

Summary

This assignment provides the opportunity to research pickling and error handling. Both can be very useful for working with collections of data. A very simple example of pickling different data types is demonstrated. Error handling is added to guide the user in inputting the desired data for the demonstration.