Katie London

11/28/2021

Foundations of Programming: Python

Assignment07

<https://github.com/KatieLondon/IntroToProg-Python-Mod07>

Creating Demonstrations of Pickling and Structured Error Handling

Introduction

In assignment 7 we create a script to demonstrate our knowledge of pickling and structured error handling. Pickling is a technique used to store data in a binary format. Structured error handling is a feature of many coding languages including Python. It allows the developer to capture potential errors from user input or errors from mathematical operations and choose how to handle them. The syntax involves using try and except blocks of code, where the code inside the try block is executed and if any exceptions are raised, it skips the rest of the commands in the try block and jumps down to execute the code in the except block. In addition to working with the built-in exceptions, Python provides the flexibility to create custom error classes.

Pickling

Pickling is a module in Python that allows you to read and write data or objects into a binary format. The binary format obscures the data, making it hard to read, but it does not make the data secure. The benefit of using a binary format is that it can reduce the file size. Pickling is one of three serialization methods in Python. The other two methods are the marshal module or json module. The pickling module contains two functions, dump and load.

I found several sites to be helpful when learning about the Python pickle module. I started with the The Python Pickle Module - How to Persist Objects in Python, (Real Python, <https://realpython.com/python-pickle-module/>, 2021) (External Site) and the Pickle in Python: Object Serialization, (Datacamp, <https://www.datacamp.com/community/tutorials/pickle-python-tutorial>, 2021) (External Site) sites, where I learned about the term serialization and the other serialization formats, marshal and json. I also consulted the Python documentation (pickle - Python Object Serialization, <https://docs.python.org/3/library/pickle.html>, 2021) (External Site) which provided the syntax and parameters for the dump and load functions.

The demo uses the dump and load methods within the pickle module. The dump method writes the serialized object to a file and the load method does the inverse. The load function unpickles data from a file and reconstitutes it as a Python object. For this demo I created a list of vegetables, then I opened a file object and wrote the pickled data to the file. Since the pickle module writes data in a binary format, we need to pass the argument ‘ab’ for append binary when creating the file object.

**import pickle #loads the pickle module**

**# Dump demo**

**lstVegetables = ['carrot', 'celery', 'broccoli', 'cabbage', 'kale']**

**objFile = open('Vegetables.abc', 'ab')**

**pickled\_vegetables = pickle.dump(lstVegetables, objFile)**

**objFile.close()**

After closing the file, I reopen the file Vegetables.abc with the ‘rb’ argument which stands for read binary and use the load method to get the data back into list form.

**# Load demo**

**lstPickled = open('Vegetables.abc', 'rb')**

**unpickled\_vegetables = pickle.load(lstPickled)**

**objFile.close()**

**print(unpickled\_vegetables)**

Structured Error Handling

I’ve created five code demonstrations of structured error handling. In the first demonstration, I show the basic functionality using only one generic except block. In the second and third demonstration I show how you can have multiple except blocks, each one capturing a certain type of error. In the fourth demonstration, I show how to create a generic except block as the last except block to catch all other errors and I also demonstrate the use of the keyword finally. In the fifth demonstration, I create a custom exception class and show how to use it by raising an exception.

In researching structured error handling in Python, I found several sites to be helpful. On the Towards Data Science site, the article “How to Define Custom Exception Classes in Python” (Toward Data Science, <https://towardsdatascience.com/how-to-define-custom-exception-classes-in-python-bfa346629bca>, 2021)(External Site) was helpful in better understanding the \_\_int\_\_() and \_\_str\_\_() methods inside custom exception classes. I also referred to the Python Error and Exceptions tutorial in the Python 3.10.0 Documentation (Python 3.10.0 Documentation, <https://docs.python.org/3/tutorial/errors.html>, 2021)(External Site) from which I learned the difference between syntax errors and exceptions. Finally, the last site I referenced was the W3 School’s Python Try Except Tutorial (W3 School, <https://www.w3schools.com/python/python_try_except.asp>, 2021)(External Site). I particularly liked the structure of the different features of the try-except blocks and modeled my demos after it.

Basic try-except Demo

For the first demonstration of the try-except block logic, the user is asked to choose a vegetable from the list to display. There are 5 elements in the list. If the user chooses a number less than 5, the corresponding fruit will be displayed. However, if the user chooses an index of 5 or greater it will throw an error because the list index is out of range. When this happens we will execute the code in the except block which is a custom error message that alerts the user to choose an index in the correct range. In addition to the custom error message, using the error object generated we can also print the error object itself, the type of error object and the docstring from the error object class function.

Multiple Except Blocks Demo

Python has many built-in exceptions. Using multiple except blocks we can specify different actions for different types of errors. In demo 2 and 3, I provide an example of using multiple except blocks, one for TypeError exceptions and one for IndexErrors exceptions. In the script, I force an index error by setting the list index, intVeg to 6 which is outside the range of the list. When the code runs, we can see that the corresponding message we get is from the IndexError except block. Next, when I change the data type of the index to be a string, I create a TypeError because a list index must be an integer. As a result, the code within the TypeError except block is printed. Having multiple except blocks and using built-in Python exceptions can be very helpful in handling user input and providing more detail about errors encountered during runtime.

Multiple Except Blocks with Except and Finally Keywords

In the fourth demo, I created a TypeError exception. I removed the TypeError except block and put a generic except block in its place. The benefit to having the generic except block at the end of the try-except block is that it will catch all other exceptions not specifically named in the previous except blocks. In addition to the generic except block, I also used the keyword “Finally” in this demo. The “Finally” keyword block always executes, regardless of whether an exception occurred or not. This can be a helpful feature for troubleshooting or providing status updates to the user as the program is running.

Raising an Exception

Another feature within structured error handling is the ability to raise an exception. By using if-else logic within the try block, you can use the keyword “raise” to raise different predefined exceptions. This is an alternative to having multiple except blocks, one for each exception. In demo 4, I raise an IndexError and capture the error object to print out details of the exception.

Custom Error Classes

In the final demo, I created a new exception class, Not\_Vegetable\_Error, which raises an exception when the list item specified is not a vegetable. This is the case for index 4 in the list which is a tomato. The custom class is created at the beginning of the structured error handling section. It contains a docstring and one function, \_\_str\_\_ and returns a custom error message.

Running Python Script

I ran the pickling demo in PyCharm. The only output was the printed list of vegetables as seen below in figure 1. I also ran the pickling demo on the Terminal as shown in figure 2. I ran the structured error handling demo, from the Assignment07 script, in PyCharm and in the Terminal. In PyCharm, I ran each of the 6 demos as shown in Figure 3 - Figure 7 below. On the Terminal I only tested one demo option since I had previously demonstrated the other were working correctly in PyCharm.

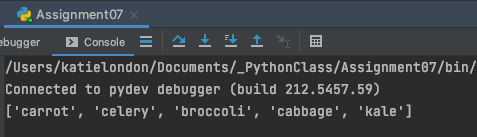


Figure 1: Pickling demo output in PyCharm



Figure 2: Pickling demo output in Terminal

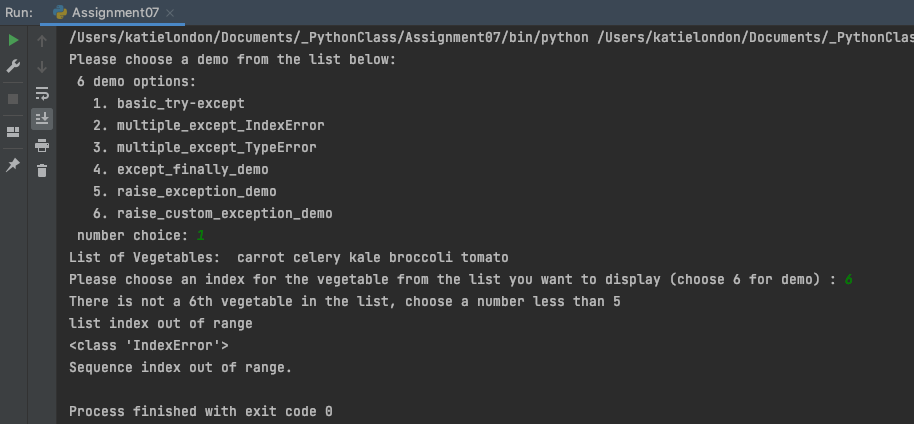


Figure 3: Screenshot of Menu option choice 1 in PyCharm

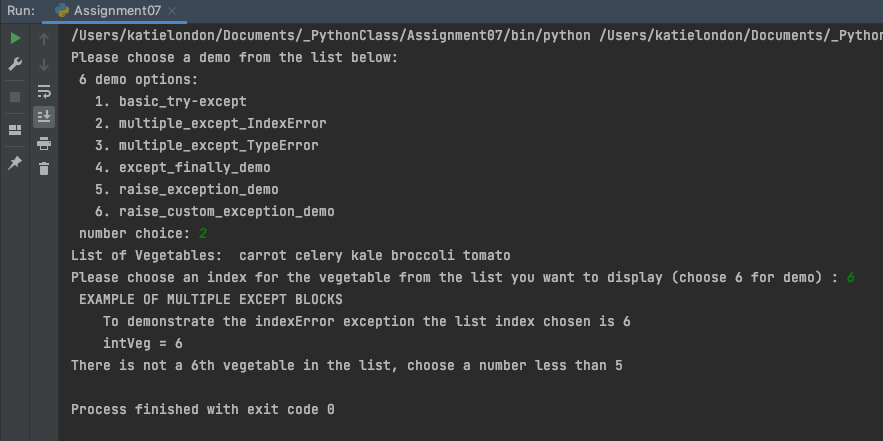


Figure 4: Screenshot of Menu option choice 2 working in PyCharm

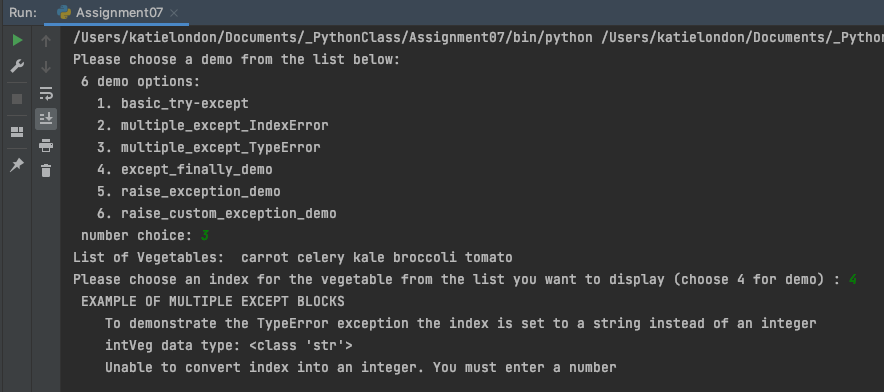
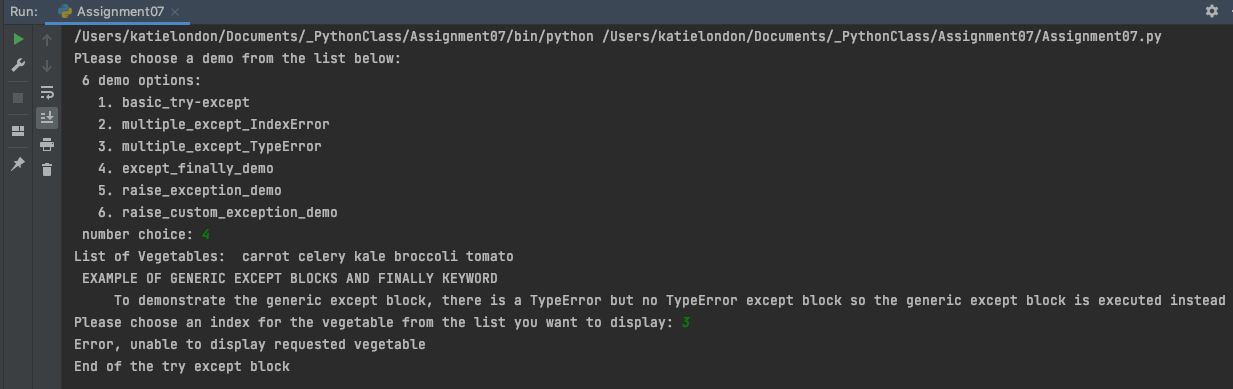


Figure 5: Screenshot of Menu option choice 3 working in PyCharm

Figure 6: Screenshot of menu option choice 4 working in PyCharm

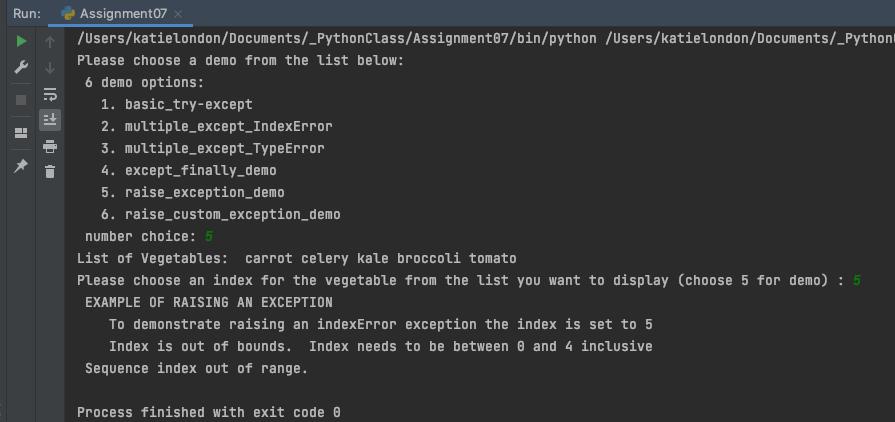


Figure 7: Screenshot of menu option choice 5 working in PyCharm

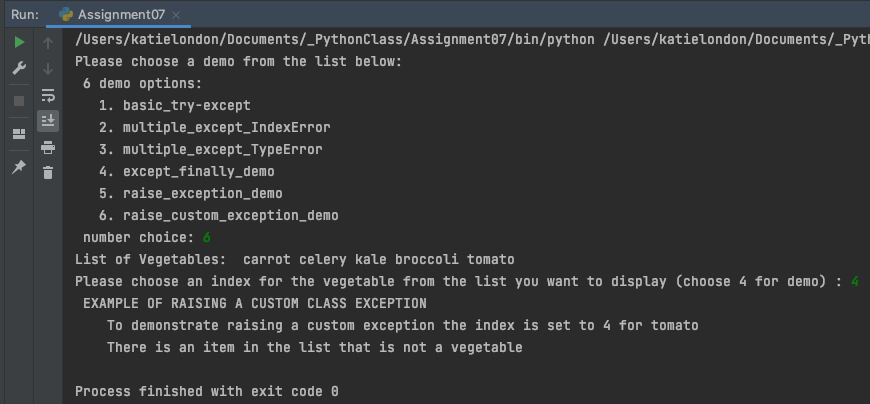
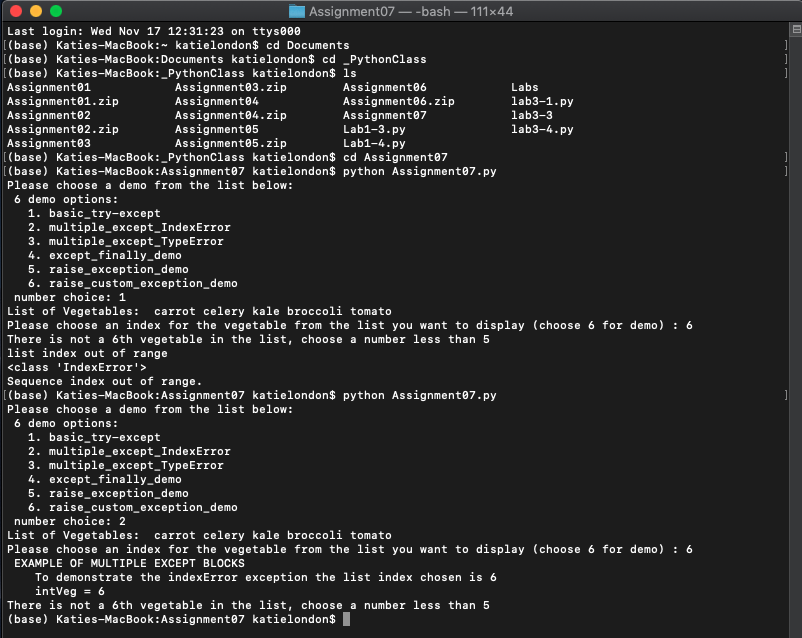


Figure 7: Screenshot of menu option choice 5 working in PyCharm

Figure 8: Screenshot of the Assignment06 script successfully running in Terminal

## Summary

To complete assignment 7, we researched and created a script with demonstrations of pickling and structured error handling. I used the internet sites listed earlier to learn about the two topics. Then I created demonstrations in the Assignment07.py script to show what I learned. For the pickling demonstration, I created a file and stored the serialized data within the file in binary format. Then I unpickled the binary data and stored it in a list. For the error handling demo, I created six test cases, to demonstrate the different error handling features.