Andrew Li
Dec. 1, 2020
Foundations of Programming: Python
Assignment07
https://github.com/idrew4u/IntroToProg-Python-Mod06

Pickling and Exception Handling in Python

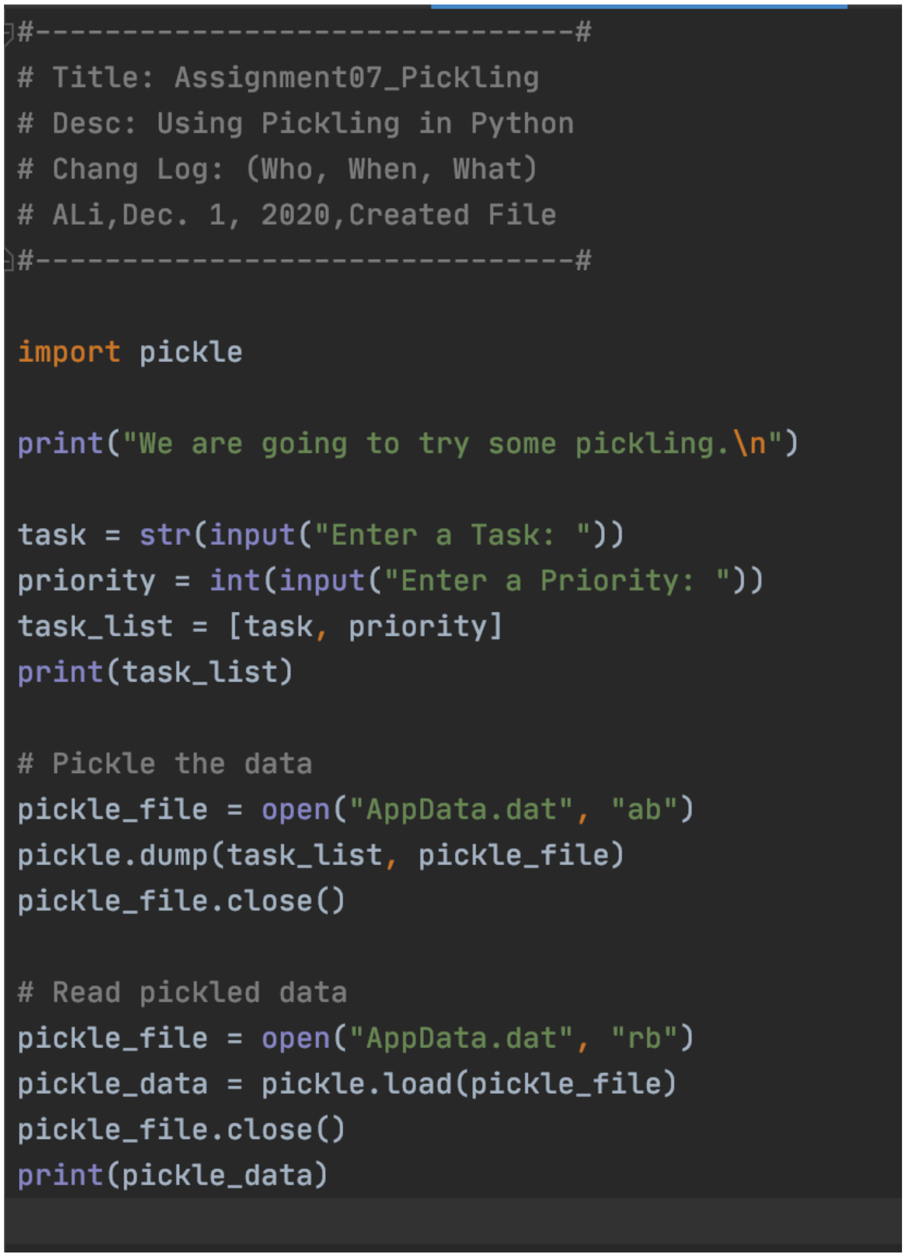
Introduction

In this assignment, we were asked to explore two concepts: pickling and exception handling. Pickling refers to serializing data or binary files. In python, exceptions are events that disrupt the code or causes errors. So when talk about exception handling, we are discussing how the script handles each exception that occurs.

Pickling in Python

During pickling, we are essentially converting data into a binary format. Pickling obscures how we read the data, making it harder to read or understand. One may use it to make it difficult for someone to just open and file and read it. Another positive of pickling is that it may also reduce the actual size of the data, so it can be beneficial when working with large amounts of data. Although it might make your data harder to read, it does not encrypt your data. When can then un-pickle the data, or deserialize the data, to covert it back to a more readable format.

In order to use the pickle module, we must first import it into our script. By using the "import" syntax, we are attempting to import the module/code from another code file. In our example, I've brought back a familiar script we were working on previously (Figure 1). After gathering the data, we want to open the file we want to save to and append/add to it. We can then store the data gathered with the pickle.dump method, which "dumps" the data into the file. When the data is stored, we can open the file again and "read" it, and have it read back to us with the pick.load method.



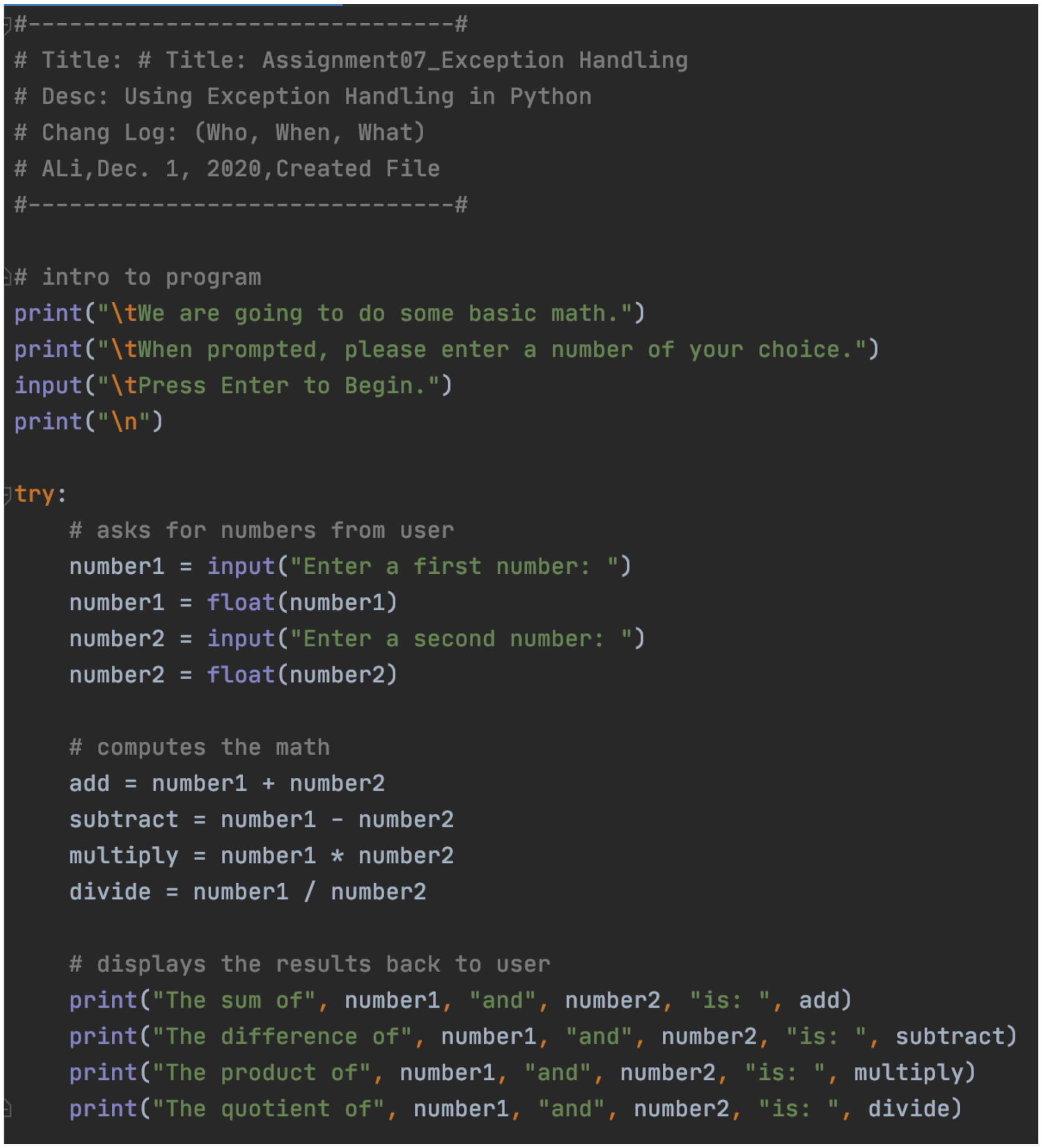
```
#------------------------------#
# Title: Assignment07_Pickling
# Desc: Using Pickling in Python
# Chang Log: (Who, When, What)
# ALi,Dec. 1, 2020,Created File
#------------------------------#

import pickle

print("We are going to try some pickling.\n")

task = str(input("Enter a Task: "))
priority = int(input("Enter a Priority: "))
task_list = [task, priority]
print(task_list)

# Pickle the data
pickle_file = open("AppData.dat", "ab")
pickle.dump(task_list, pickle_file)
pickle_file.close()

# Read pickled data
pickle_file = open("AppData.dat", "rb")
pickle_data = pickle.load(pickle_file)
pickle_file.close()
print(pickle_data)
```

Figure 1: An example of a pickling script.

Exception Handling

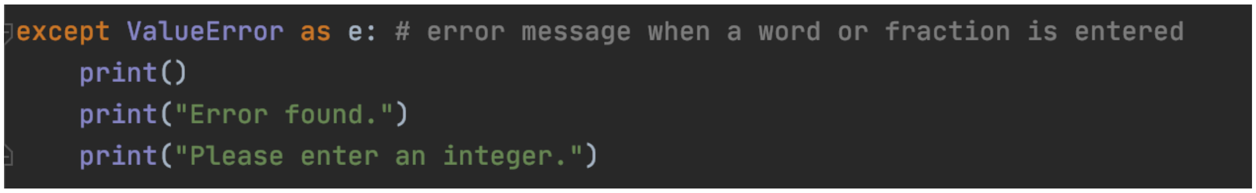
Exception handling sounds pretty close to what is actually does, it tells the script how to "handle" certain errors or events ("exceptions"). If an improper entry or input is entered, it can cause the code to prompt error messages that may be hard to decipher. These error codes may be easy for a developer to interpret, but the user may not know what that error means. It is good to use exception handles when there is human interactions that may cause potential issues. It allows us have control of how the program handles the error, and customize the messages given to the user that may help them better understand why the program has stopped. We outline these exception handles by using the try-except blocks of code.

In my example, I again, brought back an earlier script we use with basic math (Figure 2). After gathering input from the user, the code compute the math. If for any reason, the input breaks the code, for example, the user enters the spelled out variation of a number, instead of the integer itself, it would return an error message “Error found” and “Please enter an integer” (Figure 3).



```
#------------------------------#
# Title: # Title: Assignment07_Exception Handling
# Desc: Using Exception Handling in Python
# Chang Log: (Who, When, What)
# ALi,Dec. 1, 2020,Created File
#------------------------------#

# intro to program
print("\tWe are going to do some basic math.")
print("\tWhen prompted, please enter a number of your choice.")
input("\tPress Enter to Begin.")
print("\n")

try:
    # asks for numbers from user
    number1 = input("Enter a first number: ")
    number1 = float(number1)
    number2 = input("Enter a second number: ")
    number2 = float(number2)

    # computes the math
    add = number1 + number2
    subtract = number1 - number2
    multiply = number1 * number2
    divide = number1 / number2

    # displays the results back to user
    print("The sum of", number1, "and", number2, "is: ", add)
    print("The difference of", number1, "and", number2, "is: ", subtract)
    print("The product of", number1, "and", number2, "is: ", multiply)
    print("The quotient of", number1, "and", number2, "is: ", divide)
```

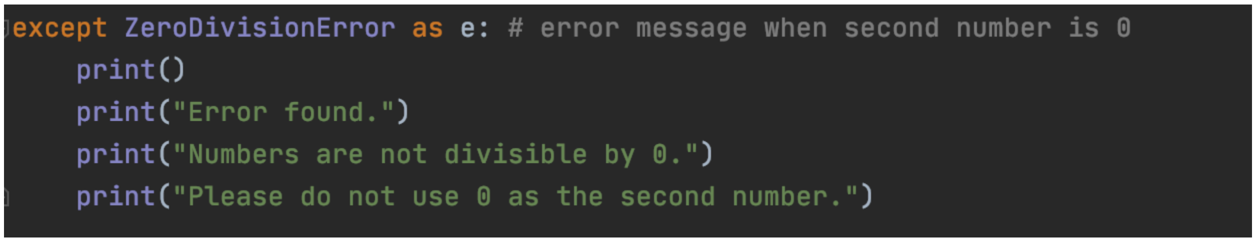
Figure 2: Starting example used to demonstrate exception handling.



```python
except ValueError as e: # error message when a word or fraction is entered
    print()
    print("Error found.")
    print("Please enter an integer.")
```

Figure 3: ValueError exception handle used.

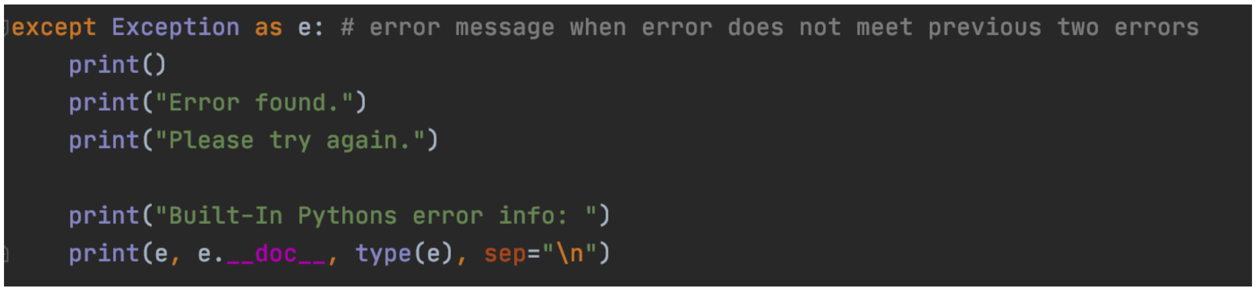
Let's say, for example, the user does enter an integer, and let's say, that integer was "0", if the user happens to enter "0" for the second number, another error message would appear. The error would not be cause by the first exception handle because what they entered is an integer and doesn't cause that particular exception. But it still returns an error. Since numbers are not divisible by "0", we get another error code, but in this case, we customize it with a message that explains why there is an error (Figure 4).



```python
except ZeroDivisionError as e: # error message when second number is 0
    print()
    print("Error found.")
    print("Numbers are not divisible by 0.")
    print("Please do not use 0 as the second number.")
```

Figure 4: ZeroDivisionError exception handle used.

If all else fails, we have the last exception that is available to catch all remaining errors that we may come across. Honestly, I am not sure if there are any other errors that could found, but if there is, the last exception should catch it and just return a simple "Please try again" error message (Figure 5).

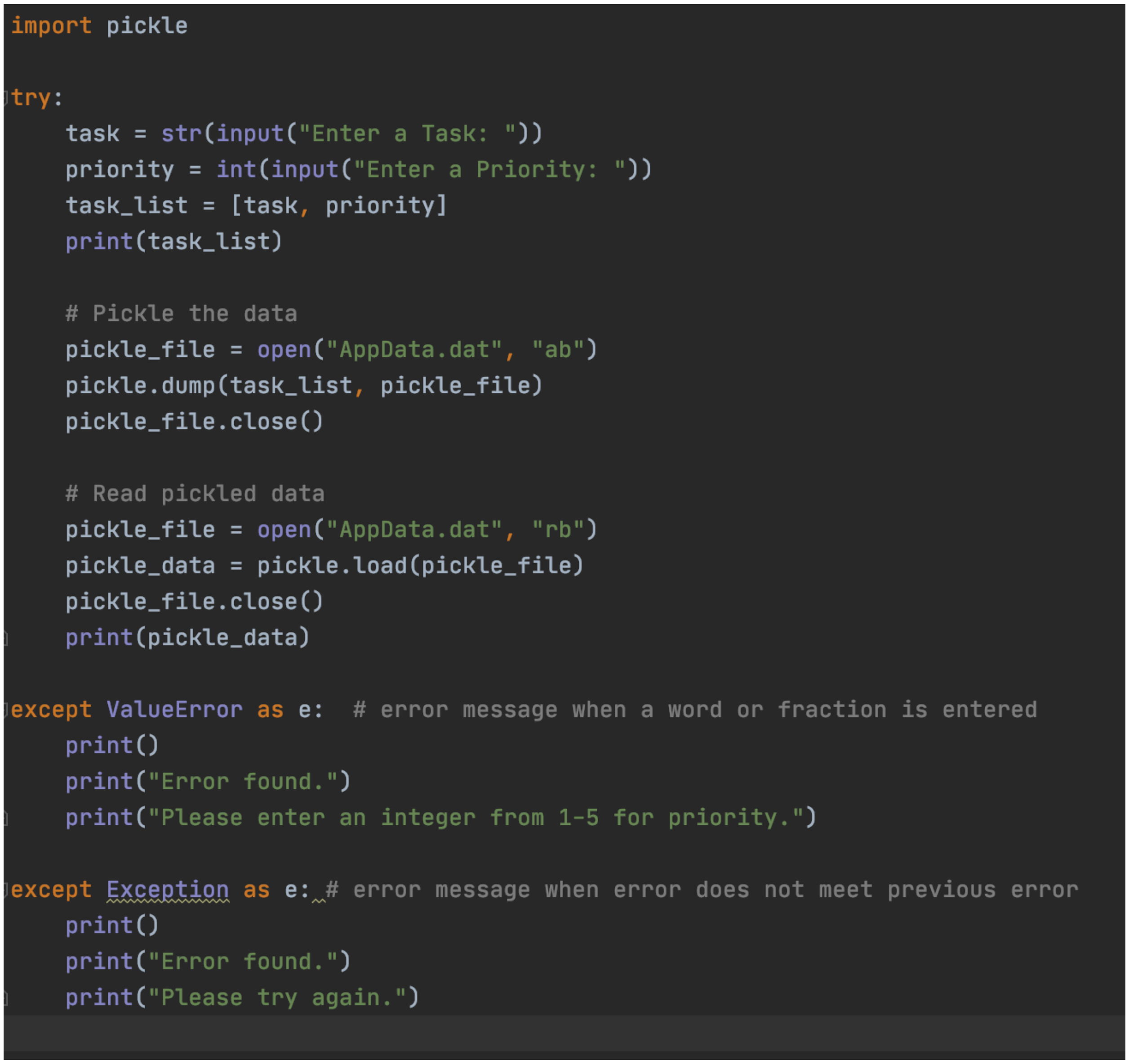


```python
except Exception as e: # error message when error does not meet previous two errors
    print()
    print("Error found.")
    print("Please try again.")

    print("Built-In Pythons error info: ")
    print(e, e.__doc__, type(e), sep="\n")
```

Figure 5: If the previous two exceptions do not meet the criteria, this would be the last exception handle used to catch any other remaining errors.

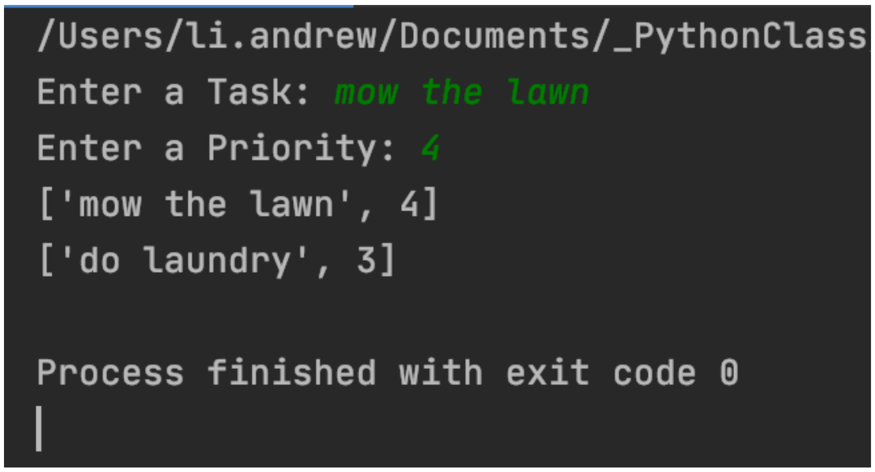
Running the Program

I was able to combine use aspects of the two new concepts and put them into one script (Figure 6.) After creating the script, we verified the program is working properly by running it in PyCharm (Figure 7) and the terminal command window on macOS (Figure 8). I also included an example of what the file looks like after being pickled (Figure 9).



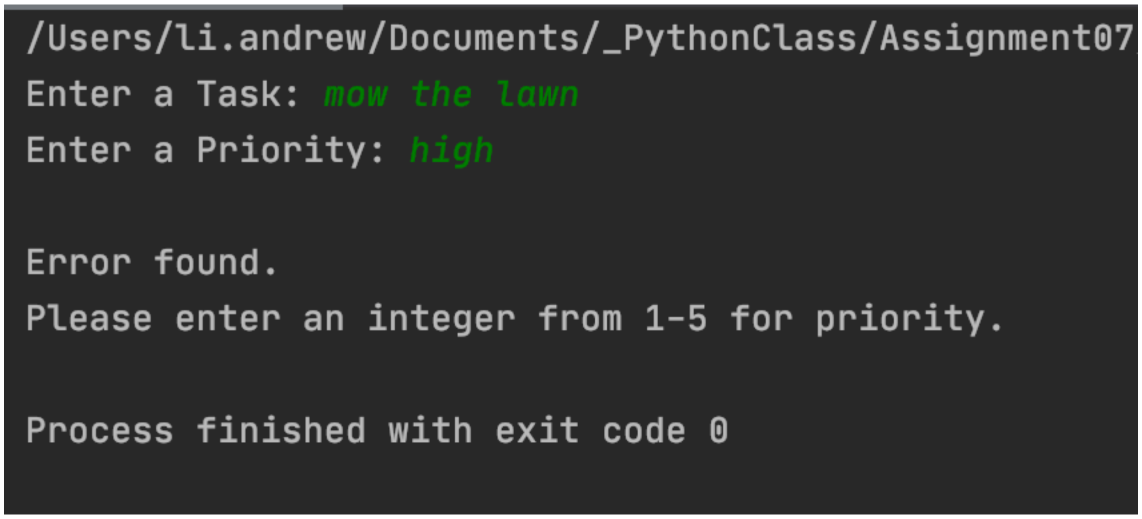
```python
import pickle

try:
    task = str(input("Enter a Task: "))
    priority = int(input("Enter a Priority: "))
    task_list = [task, priority]
    print(task_list)

    # Pickle the data
    pickle_file = open("AppData.dat", "ab")
    pickle.dump(task_list, pickle_file)
    pickle_file.close()

    # Read pickled data
    pickle_file = open("AppData.dat", "rb")
    pickle_data = pickle.load(pickle_file)
    pickle_file.close()
    print(pickle_data)

except ValueError as e:  # error message when a word or fraction is entered
    print()
    print("Error found.")
    print("Please enter an integer from 1-5 for priority.")

except Exception as e: # error message when error does not meet previous error
    print()
    print("Error found.")
    print("Please try again.")
```

Figure 6: Script including both concepts of pickling and exception handling.



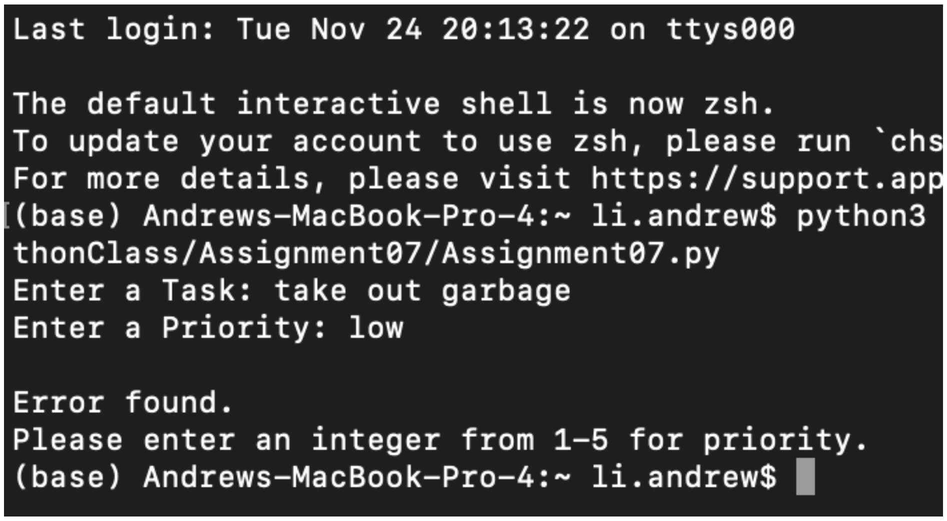
```
/Users/li.andrew/Documents/_PythonClass
Enter a Task: mow the lawn
Enter a Priority: 4
['mow the lawn', 4]
['do laundry', 3]

Process finished with exit code 0
```

Figure 7a: Program run in PyCharm with no errors.



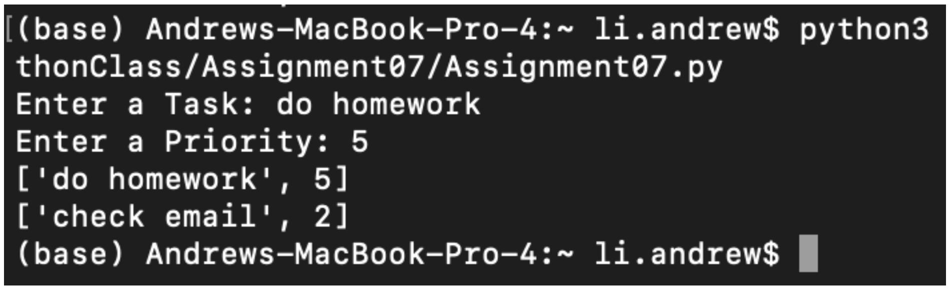
```
/Users/li.andrew/Documents/_PythonClass/Assignment07
Enter a Task: mow the lawn
Enter a Priority: high

Error found.
Please enter an integer from 1-5 for priority.

Process finished with exit code 0
```

Figure 7b: Program run in PyCharm with errors.



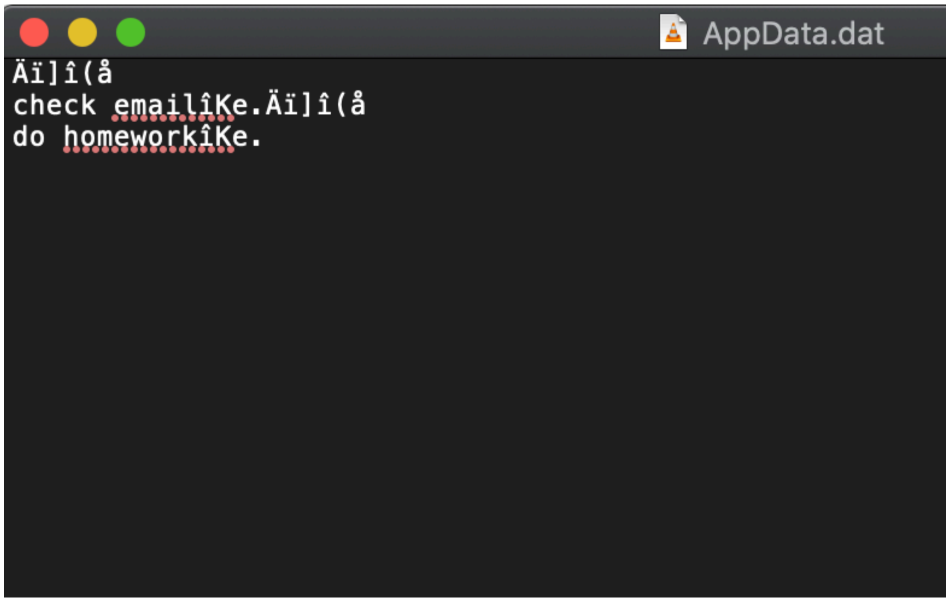
```
Last login: Tue Nov 24 20:13:22 on ttys000

The default interactive shell is now zsh.
To update your account to use zsh, please run `chs
For more details, please visit https://support.app
(base) Andrews-MacBook-Pro-4:~ li.andrew$ python3
thonClass/Assignment07/Assignment07.py
Enter a Task: take out garbage
Enter a Priority: low

Error found.
Please enter an integer from 1-5 for priority.
(base) Andrews-MacBook-Pro-4:~ li.andrew$
```

Figure 8a: Program run in macOS terminal with errors.



```
(base) Andrews-MacBook-Pro-4:~ li.andrew$ python3
thonClass/Assignment07/Assignment07.py
Enter a Task: do homework
Enter a Priority: 5
['do homework', 5]
['check email', 2]
(base) Andrews-MacBook-Pro-4:~ li.andrew$
```

Figure 8b: Program run in macOS terminal without errors.



```
Äï]î(å
check emailîKe.Äï]î(å
do homeworkîKe.
```

Figure 9: Sample of “AppData.dat” file after being pickled.

Summary

We were able to demonstrate two new concepts we've learned in python: pickling and exception handling. We used pickling in order to obscure the data collected and convert into binary format. We also took the idea of exception handling to customize error messages to the user. After we are able to understand the concept a little better, we were able to combine the two in a script and validate the pickling of the file.