David Grunloh

May 25, 2021

IT FDN 110 A

Assignment 07

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

Pickling and Structured Error Handling

# Introduction

In Assignment 07, I created a Python application which utilized Pickling as well as Structured Error Handling This application is used to gather brews and their cost from a user and store that data in a pickle dat file. The functionality of the application is mostly the same as some of the previous applications, but with the addition of the capability to store the data in a pickle as well as utilizing error handling. This code is broken into functions making the code easier to manage and reuse in the future.

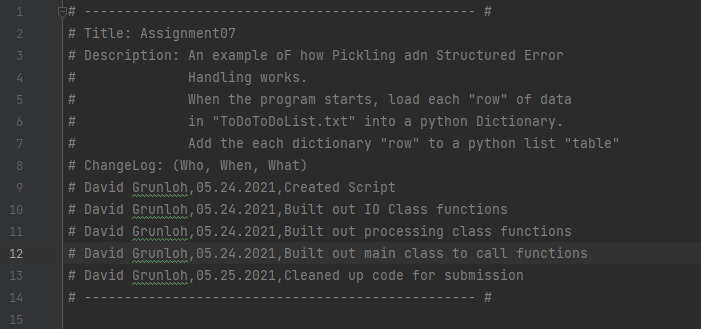
# **Writing the script**

In the script for this assignment there are 6 sections of code which inclues import modules, header, define variables, processor class, IO class, and main. These will be described in more detail below as they are used within the application.

## Header

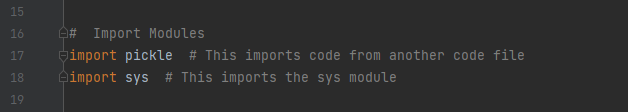
This portion of the script is used to provide information about the purpose of this script to anyone who needs to review or edit the script in the future. The main components that should be included are title, description, author, and change log.

As you can see in Figure 1 below, the application was created by RRoot, but I made several updates and changes to this application. Each of those adjustments were documented in the change log portion of this header.



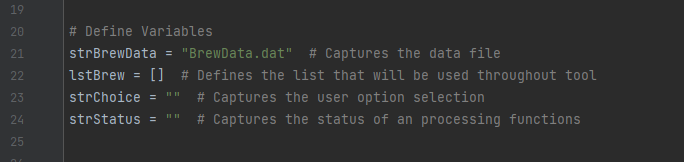
***Figure 1***

## Import Modules

In this section of code, I import the pickle module as well as the sys module. These modules are imported to utilize code that is included in another application. These particular modules are standard prebuilt modules that are included in the software. You can see below in figure 2 that importing the modules is a pretty simple command.   
***  
Figure 2***

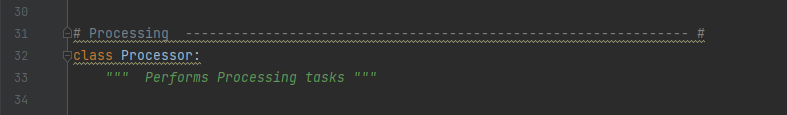
## Define Variables

In this section of code, I define some variables that will be used throughout the application. It is a good practice to define these at the beginning as shown in figure 3 for ease of use.



***Figure 3***

## Class Processor

This class, which you can see defined in figure 4 below includes the functions that do all the processing for the application. Within this class there are several different functions which complete task including read data from file, add data to list, remove data from list, and write data to file. 

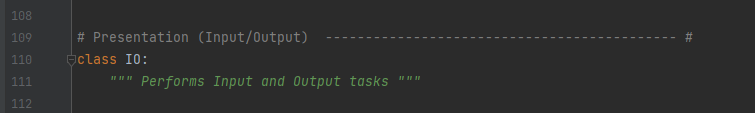
***Figure 4***

## Class IO

The IO Class, which you can see defined in figure 5 below includes the functions that do all the input output operations that interact with the user. Within this class there are a couple different functions which complete task including:

* print\_current\_brews
* input\_new\_brew

The input\_new\_brew function will be discussed further below in the next section as it contains an example of structured error handling.



***Figure 5***

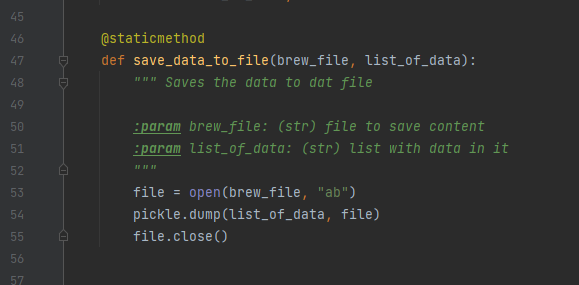
## Main

For Assignment07, I will not walk through each step of the application, but will focus on the functions and their calls which include the features of this assignment, Pickling and Structured error handling. This application is built to be pretty simple, but with the basics around building out a larger application where these pieces of functionality would provide greater value. I will be covering the 3 functions, save\_data\_to\_file, read\_data\_from\_file, and input\_new\_brew below.

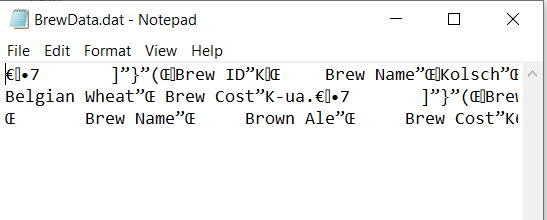
### Pickling

In the functions save\_data\_to\_file and read\_data\_from\_file, we utilize pickling which allows us to store complex data in a file with a single line of code.

In figure 6 below, you can see that we are able to store a list object, list\_of\_data in the data file as a list by calling the pickle.dump command in the save\_data\_to\_file function. The ability to store these more complex objects may not be necessary for simple applications such as this, but when building out larger, more complex applications, this functionality would be extremely helpful. You can access and manipulate the contents of these more complex objects right from the file they are hosted in.

  
***Figure 6***

In the image below in figure 7, you can see that when the data is saved to the file, it is not stored in plain text. When the data is stored, it is obscured and the storing it in the binary format can also reduce the file size. You can also notice that the data is not fully masked and is just obscured, so this shouldn’t be used to mask data.



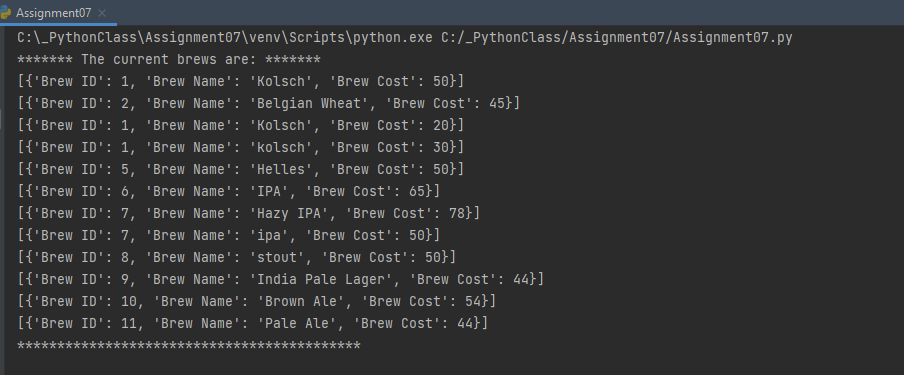
***Figure 7***

In order to access the data, I utilized the read\_data\_from\_file function. Within this function, because there were multiple rows in the list, I utilized a while loop to loop through each row of the list that is stored within the Pickle storage object and append that to a list. At the end, I utilized the EOFError to capture the error of hitting the end and then break to get out of the loop. You can see pulling data from the pickle storage object is pretty straightforward in figure 8.



***Figure 8***

When you print the contents that are pulled from this storage object, you can see that it maintained its structure based on how it was entered into the file. See figure 9 below which shows the output displayed within PyCharm.



***Figure 9***

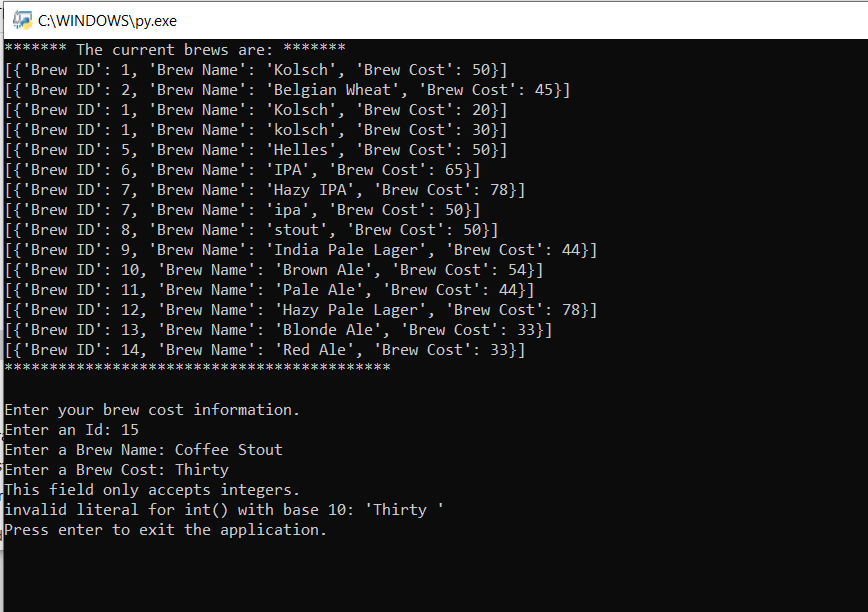
### Structured Error Handling

In the below section of code shown in Figure 10, you can see code that was used to provide structured error handling for end users when entering new brews. You can see that structured error handling was used with a try statement with an except clause. The try statement has the user input a brew id, where it is expecting an integer. The except cause in this case is designed to capture a value error which would occur if the user entered something that is not an integer. When this happens, it provides a nice error message stating that the field is expecting an integer, as well as the python specific error message. It then exits the application using the sys.exit() module. The same functionality was used for the brew\_cost input.



***Figure 10***

Below is figure 11 that shows the structured error handling at work form the command prompt. You can see that when I typed in Thirty for Brew Cost, you received an error stating that this field only accepts integers. It also provides the Python specific error. It then pauses to allow you to see the error from the command prompt.

******

***Figure 11***

# Summary

In this assignment, I utilized python functions in addition to variables and list to gather input from a user and process it into a Pickle storage object. This application was built with basic functionality in mind and is a start to a larger application with greater functionality. I also utilized structured error handling to help the user understand what error occurred when they enter an invalid option into an input field. These python features would provide a significant amount of valid in a more complex application.