Name: Stephanie Tarczynski

Date: 12/1/2020

Course: IT FDN 110 A

Assignment: Module 07, Assignment 07

GitHub Link: <https://github.com/steptar/IntroToProg-Python-Mod07.git>

How to Catch Errors and Use Pickling in Python

# Understanding the Task

Before creating the script, I made sure I understood the task at hand. Assignment 07 requires the developer to create their own demo to demonstrate Python error handling and pickling. Extra research was conducted via the internet to find more information on the concepts taught in Module 7. The assignment requires this code to be run in PyCharm and the OS console window.

# Creating the Script

## Part 1: Error Handling

To demonstrate error handling in Python, I knew I wanted to cover a few different methods for catching errors, custom errors, and standard errors. I wanted to keep the program goal simple so that the error handling purpose would be easy to understand. So, I decided to create a program that helps the user perform basic division to distribute snacks evenly amongst a group of people. The program starts by collecting the number of people and the number of snacks available from the user as shown in Figure 1. These values are passed to the SnackDivider() function

*Figure 1. Initial data collection*

intPeople = int(input(**"Please enter the number of people: "**))  
intSnacks= int(input(**"Please enter the number of snacks available: "**))  
SnackDivider(intPeople, intSnacks)

The SnackDivider() performs the very basic function of dividing the number of snacks over the number of people, but it also includes code to catch various errors along the way. The division is done within a “try: “ block and includes if/else statements to “raise” custom errors and uses “except:” to capture the custom errors and a standard Python exception. The script for SnackDivider is shown in Figure 2 and each error is labeled and will be explained individually.

Error 1: Type Error using if/else and raise:

This program chooses to raise an error if the snacks cannot be divided evenly amongst all people. In order to catch this error, the program tries to divide the snacks amongst the number of people. An if/else statement is used to capture whether or not the remainder of the division is 0. If the remainder is not 0, the “raise:” statement is used to raise a TypeError. The TypeError is raised because the result of the division is not an integer and therefore the snacks cannot be divided evenly. Using an if/else statement creates a more customized approach to executing the TypeError.

Error 2: ZeroDivisionError:

Error 2 uses only an except statement to catch a standard error in Python- the ZeroDivisionError. If there are no people, the snacks cannot be divided amongst them.

*Figure 2. SnackDivider() function with errors numbered 1-2.*

24

1

def SnackDivider(numPeople, numSnack): *#This function's purpose is to divide the snacks evenly amongst all people* try:  
 snackPerPerson= numSnack/numPeople  
 if numSnack%numPeople != 0: *#if/else statement to raise an error* raise TypeError  
 else:  
 pass  
 except TypeError:  
 print(**"The snacks cannot be divided evenly."**) *#The snackPerPerson result was not an integer* except ZeroDivisionError: *#standard Python exception error* print(**"Zero Division Error: No people, no snacks needed"**)  
 finally:  
 print(**"Each person can have "**, snackPerPerson, **" snacks!"**)

The “finally” statement executes whether an error is caught or not. So, even if the snackPerPerson value is not an integer, the program will still tell the user the float value of how the snacks can be divided evenly. This portion of the script demonstrates basic arithmetic errors that can be caught and how multiple errors can be captured in one “try:” block.

## Part 2: Pickling

In order to combine the error handling and pickling demonstrations into one program I drew the purpose of Part 2 from the inputs of Part 1. This portion of the code collects the name and number of snacks each person received and saves this to a dictionary. The dictionary is saved to and extracted from a binary file using pickle.

Firstly, “pickle” is imported into Python. strFileName is declared as .dat file to store binary data. lstSnacks is declared to initialize the list that will store the dictionary rows.

The main portion of the code uses a while loop that executes per the number of people the user specifies in Part 1 of the program. It collects the name and number of snacks that person received and appends it to the lstSnacks. lstSnacks and the file are passed to save\_data\_to\_file() and the file is read by the read\_data\_from\_file() function. The main portion of Part 2 is shown in Figure 3.

The save\_data\_to\_file() function open the file in “wb” mode which means it will write to a binary file. The pickle.dump() function is used to “dump” the lstSnacks data into the file in a binary format. The function is shown below in Figure 4.

*Figure 4. Using python pickling to save data to a binary file*

def save\_data\_to\_file(file\_name, list\_of\_data):  
 objFile = open(file\_name, **"wb"**)  
 pickle.dump(list\_of\_data, objFile)  
 objFile.close()  
 print (**"Success"**)  
 pass

*Figure 3. Main body of part 2 to collect user input for the name/snack dictionary*

*# Main Part 2-------------------------------------------- #*print(**"Now we will store the snack data in a binary file."**)  
  
lstSnacks = []  
i=0  
while i < intPeople: *#while loop iterates until each person's snack amount is captured* str\_Name = input(**"Enter the name: "**)  
 str\_Snacks = input(**"Enter the number of snacks: "**)  
 row = {**"Name"**:str\_Name, **"Snacks"**: str\_Snacks}  
 lstSnacks.append(row)  
 i+=1  
  
save\_data\_to\_file(strFileName, lstSnacks) *#Store the list into a binary file*read\_data\_from\_file(strFileName) *# Read the data from the file into a new list object and display the contents*

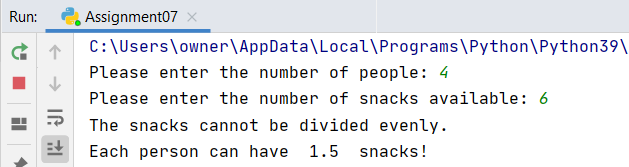
The read\_data\_from\_file() function opens the .dat file in “rb” mode which means it reads from a binary file. A while loop is used to extract the dictionary rows using pickle.load() and appending them to (“objFileData”). The loop uses a try/except block and the EOFError to collect data as long as the “End of File” error does not occur. Once the loop reaches the end of the file, objFileData is printed. This function is shown below in Figure 5.

*Figure 5. Using python pickling to save data to a binary file*

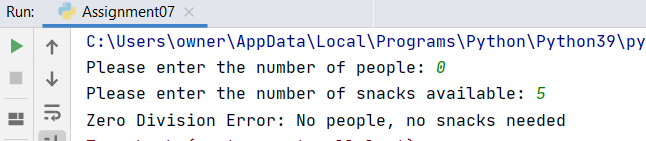
def read\_data\_from\_file(file\_name):  
 objFile = open(file\_name, **"rb"**)  
 objFileData = []  
 while True:  
 try:  
 objFileData.append(pickle.load(objFile))  
 except EOFError:  
 break  
 objFile.close()  
 print(objFileData)

# Running the Code

To run the code, I first tested Part 1 (Error Handling) in Pycharm. Figure 6 demonstrates the program catching Error 1 (Type Error) where the snacks cannot be easily divided. Figure 7 catches Error 2 (ZeroDivisionError) where there are no people to divide the snacks amongst.

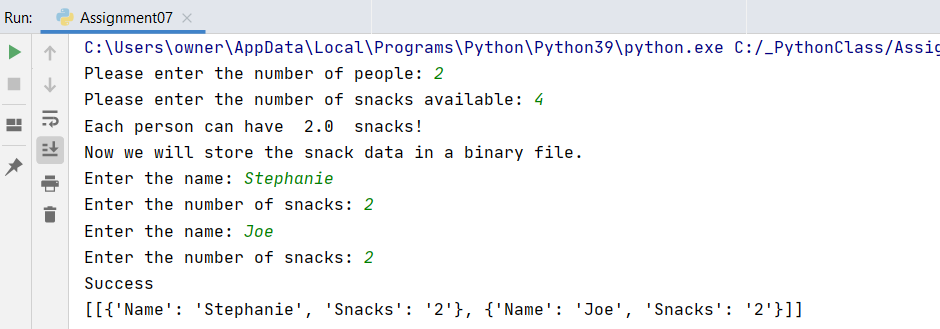


*Figure 6. Error 1 (TypeError)*

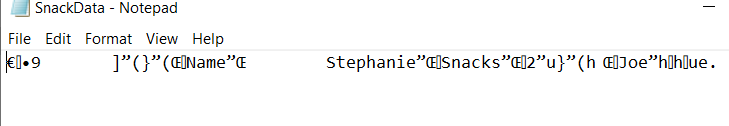


*Figure 7. Error 2 (ZeroDivisionError)*

After testing the error, I entered value for the number of people and number of snacks that would not return any errors. I then moved onto Part 2 of the program. I entered the names and number of snacks each person had. The program stored and extracted this dictionary of data to a binary file and extracted and printed the results. This full fun is shown in Figure 8. An image of the binary file is shown in Figure 9.

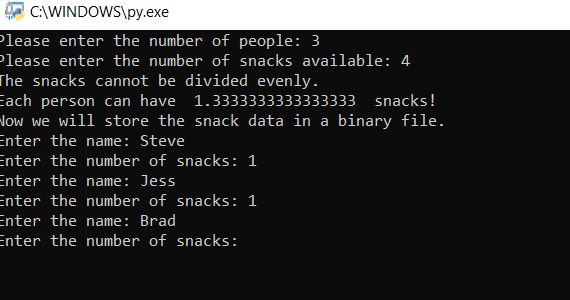


*Figure 8. Full Program Run*

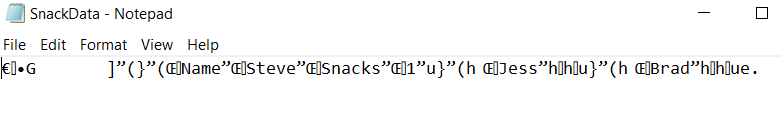


*Figure 9. Binary File*

After testing the code in PyCharm, I then ran the same program in the command console to verify it can be run from there as well. The program can be seen running in Figure 10 and the binary file can be seen in Figure 11.



*Figure 10. Assignment07 running in the console window*



*Figure 11. SnackData.dat output from running Assignment07.Py in OS console window*

# Summary

To complete this assignment, I performed additional research on the internet to get more information on error handling and pickling. I created a program that demonstrated both concepts in one script. For Part 1 of the program I performed a basic division operation and used a if/else statement to raise an error and used a standard error. Part 2 of the program stored and extracted binary data using pickle.dump() and pickle.load().

# Resources

Try/Except Resources:

1. <https://www.tutorialspoint.com/python/python_exceptions.htm>

This website goes over the basics of error handling and gives a list of the standard python Standard Exceptions. I used this website to find the basic arithmetic error functions I wanted to use for handling snack division errors

1. <https://www.tutorialsteacher.com/python/exception-handling-in-python>

This website explains things a bit more simply than the first. This gave me a good understand of how to use standard errors using a simple try/except block. It also demonstrated how to use if/else statements to raise an error.

Pickle Resources:

1. <https://www.geeksforgeeks.org/understanding-python-pickling-example/>

This website showed how to import and extract a dictionary using Python pickling.

1. <https://www.kite.com/python/answers/how-to-read-a-pickle-file-in-python>

This website combined both lessons. It showed how to append multiple rows in a pickle file to one object. It uses a while loop and try/except block that executes until the loop reaches an EOFError (end of file). I ultimately used this in my code because it was easier for me to understand and cool that it utilized both concepts.