

Mark Brugger
November 30, 2022
IT FDN 110 A – Foundations of Programming: Python
Assignment07
<https://github.com/MBrugger11/IntroToProg-Python-Mod07>

DIY Demo Script

Introduction

This week it was our turn to create a demo script to showcase concepts we learned this week. It was our first time in a few weeks starting with a blank file rather than some starter code.

Learning concepts this week

First, we were formally introduced to interfacing with text files in Python. We were given a few snippets of code early in the class to get a feel for how text file reading/writing worked, but now the concept was expanded in scope.

Previously we used a for loop and a `row.split` command to access data from a text file. This week we learned about the `readline()` function. As one would expect, it reads one line of data. By pairing it with an while loop, it is possible to get additional lines out of the file.

The `readlines()` function (notice the s) reads all the lines at once and returns them as a list. By contrast, the `read()` function returns a string.

New this week was reading and writing from binary files instead of the text files. The function in Python that works with binary files is ***pickle*** and this process is called pickling. It is a topic that we were tasked with researching. Afternerd, also known as Karim, (see <https://www.afternerd.com/blog/python-pickle/>) has a webpage that goes into detail on what pickling is and how you can use it in your Python scripts.

The advantage to a binary file versus a text file is it is a more efficient way to store data. If you are working with a large data set, this will result in storage savings and if the file is being transmitted, it can be more quickly sent. Since it is not plain text, it keeps casual eyes away, although it is possible for a determined person to extract the data. So do not think of it as a security measure.

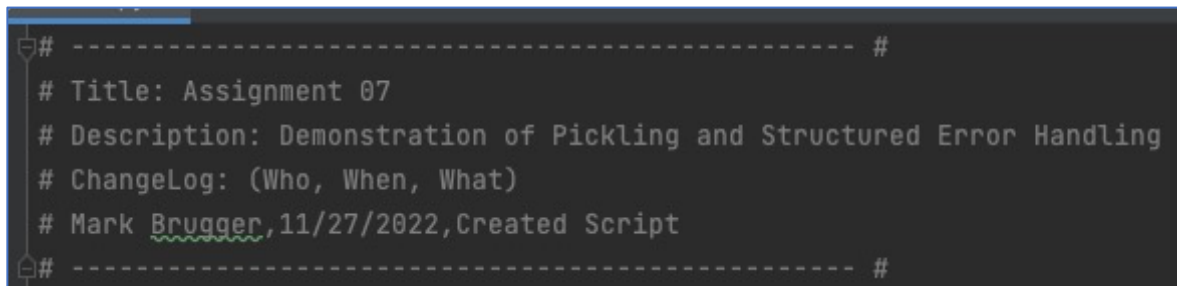
Structured Error Handling is another item that we scratched the surface with a few weeks ago but now took a deeper look. Initially we would use a simple try/except loop where we would embed some code inside the try section. If an error occurred, Python would simply stop executing code in the try section and move down to the except section and run the code there.

It is also possible to catch specific exceptions (e.g. dividing by zero errors, trying to read external files that do not exist, etc.) and write except sections for each one so that the user can get a helpful error message or specific code to help with the situation. If the specific exceptions do not catch the error, there is a catch-all “except exception” at the end for all other error types.

Process to make program

This week’s assignment was very open-ended. We needed to demonstrate working with binary files and error handling. It was recommended to demo them separately and then have a demo to demo them together.

I started with the typical header:

A screenshot of a code editor showing a script header. The text is as follows:

```
# ----- #  
# Title: Assignment 07  
# Description: Demonstration of Pickling and Structured Error Handling  
# ChangeLog: (Who, When, What)  
# Mark Brugger, 11/27/2022, Created Script  
# ----- #
```

Figure 1 - Script Header

Since we are going to be dealing with binary data, it is necessary to import the pickle function as shown in Figure 2:

A screenshot of a code editor showing the import statement for the pickle module. The text is as follows:

```
import pickle #importing the pickling function for storing data in binary format
```

Figure 2 - Pickle function

Next, we had the data section of the script (see Figure 3) to initialize variables. To be honest, this was not used as extensively as it could have been.

```
# -- Data -- #  
str_file_name = 'AppData.dat'  
lst_animal = []
```

Figure 3 - Data section of script

The processing section contains two functions. For the pickling demonstration, it will make use of the read_data_from_file and save_data_to_file functions. They read and write data respectively to an external binary file.

```
# -- Processing -- #  
def read_data_from_file(file_name):  
    """ Reads data from a file into a list of dictionary rows  
  
    :param file_name: (string) with name of file:  
    :return: (list) of dictionary rows  
    """  
  
    file = open(file_name, "rb")  
    file_data = pickle.load(file)  
    file.close()  
    return file_data  
  
def save_data_to_file(file_name, animals):  
    """ Writes data from a list of dictionary rows to a File  
  
    :param file_name: (string) with name of file:  
    :param animals: (list) you want filled with file data:  
    :return: (list) of dictionary rows  
    """  
  
    file = open(file_name, "wb") # wb is used instead of the typical w because we are working with a binary file  
    pickle.dump(animals, file)  
    file.close()  
    print("\n" + "*" * 25)  
    print("Your data has been saved!")  
    print("*" * 25)  
    return animals
```

Figure 4 - Processing section with its custom functions

The presentation section of the script is a bit of a misnomer because it not only presents information to the user and seeks input, but also does a few operations.

First, the script showed a simple try/except loop (see Figure 5) for when an external file does exist.

```
## -- Presentation (I/O) -- #

# The first part of the main section of the script will go over simple error handling
#
# obj_file = open("readme.txt", "r")
# If the comment # is removed from the line above and readme.txt does not already exist in the local folder
# Python will fail due to a runtime error (No such file or directory).
#
# Instead, a simple try/except loop is used to catch this error
try:
    obj_file = open("readme.txt", "r")
except:
    print("\n" + "*" * 80)
    print("readme.txt does not currently exist. It will be created by the program later.")
    print("*" * 80)
```

Figure 5 - Example of simple try/except error handling

Next, the script asked (shown below in Figure 6) for two pieces of information from the user. Using pickling, that information was stored in a binary file.

```
# Next, the script will show a simple demonstration of pickling
# Pickling saves data in binary format, rather than in a text format
# The binary file (without some work) is not readable outside of Python

str_name = input("\n\tEnter the name of an animal: ")
int_quantity = int(input("\tEnter the quantity of this animal: "))

# Now the information will be combined into a list

lst_animal = [str_name, int_quantity]

# Using the save_data_to_file function, the list is placed in a binary file.

save_data_to_file(str_file_name, lst_animal)

# Next, the binary file will be read into memory

lst_file = read_data_from_file(str_file_name)
print("\n\b")
print(lst_file)
```

Figure 6 - Simple example of pickling

Finally, the script went through pickling while checking for errors. The situation was a bit contrived, but by having the script load the binary file at the beginning, it allowed an opportunity for the file not to exist and make use of the except FileNotFoundError section.

```
try:
    obj_file = open("AppData23.dat", "rb")

    obj_file_data = pickle.load(obj_file)

    int_first = int(input("\n\tEnter a number: "))
    int_second = int(input("\tEnter a second number: "))

    str_math = str(23 * int_first + 11.5 / int_second)

    print("\nNow we will solve the equation  $23 * X + 11.5 / Y$ , where X is your first number and Y is the second.")

    print("The answer is " + str_math)

    print("\nThe last time the script was run, the following data was saved: ")
    print("The first two numbers were the numbers chosen and the last number is the answer to the math operation.")

    print(obj_file_data)

    lst_nums = [int_first, int_second, str_math]

    obj_file = open("AppData23.dat", "wb")

    pickle.dump(lst_nums, obj_file)

    obj_file.close()
```

Figure 7 - Pickling within a try section

The except sections (see Figure 8) looked for two specific types of errors: the binary file that Python is trying to load not existing, and the script trying to divide by zero due to the choice of number made by the user. Finally, for an error other than those two, there was a general except exception section at the end to give the user a friendly message and show them what Python found.

```
except FileNotFoundError as e: # e is a generic variable. Anything can be used, but typical to use an e for errors
    # This first exception is checking to see if the data file we wish to open already exists. If it does not,
    # Python will have a run-time error and terminate unexpectedly with an error message that isn't user friendly.

    print("\nFile \"AppData23.dat\" must exist before running this code.")
    print("Built in Python error info: ")
    print(e, e.__doc__, type(e), sep="\n")

except ZeroDivisionError as e:
    # This second exception is for situations where the user's choice of a second number is causing Python
    # to divide by zero

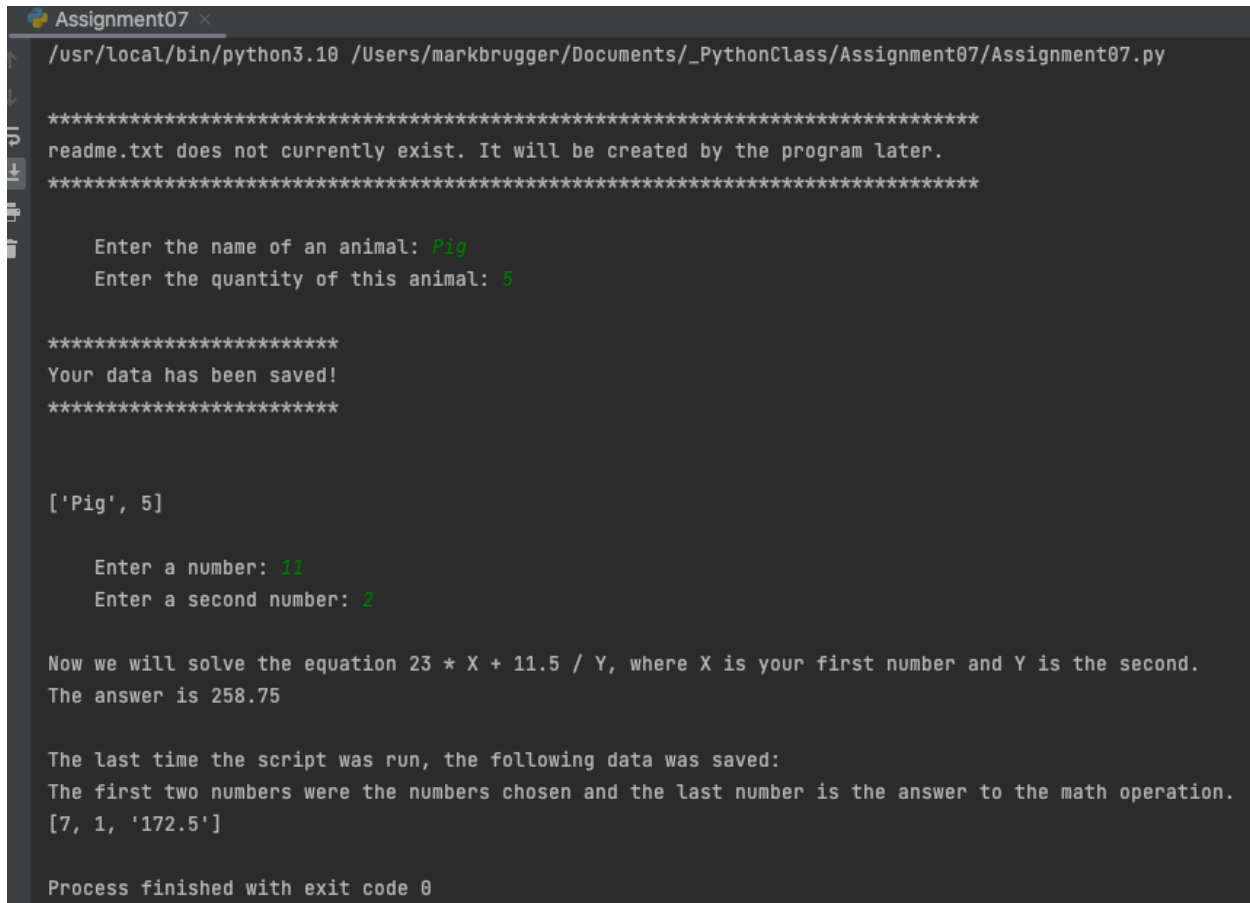
    print("The selection of 0 as your second number has caused the script to divide by zero, which is not allowed.")
    print("Built in Python error info: ")
    print(e, e.__doc__, type(e), sep="\n")

except Exception as e:
    # This final exception is a catch-all for all the other errors that didn't fall into the two categories above
    # You can write as many exception categories as you want to have as many specific error responses as possible
    # but this script demo will limit itself to three.

    print("There was a non-specific error!")
    print("Built in Python error info: ")
    print(e, e.__doc__, type(e), sep="\n")
```

Figure 8 - Except section for error handling

In Figure 9, we see an example of the script running in the PyCharm IDE.



```
Assignment07 x
/usr/local/bin/python3.10 /Users/markbrugger/Documents/_PythonClass/Assignment07/Assignment07.py

*****
readme.txt does not currently exist. It will be created by the program later.
*****

Enter the name of an animal: Pig
Enter the quantity of this animal: 5

*****
Your data has been saved!
*****

['Pig', 5]

Enter a number: 11
Enter a second number: 2

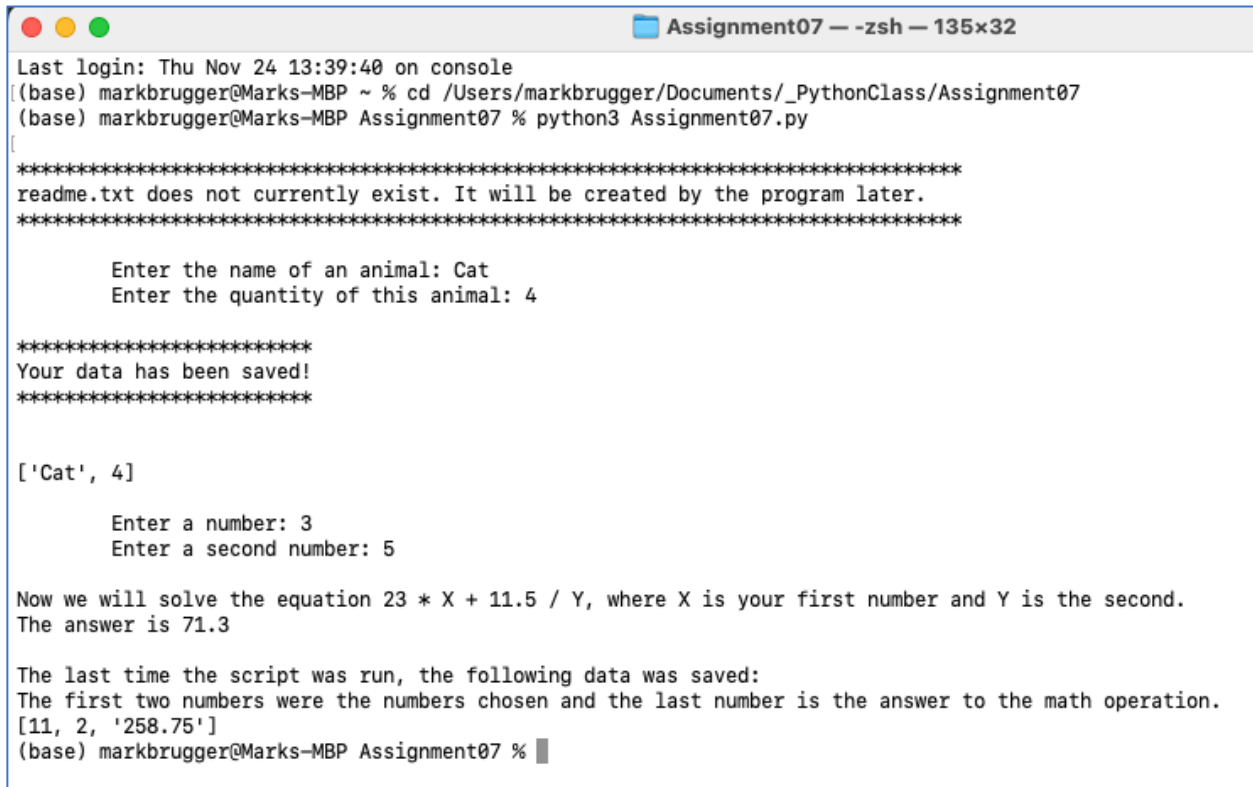
Now we will solve the equation  $23 * X + 11.5 / Y$ , where X is your first number and Y is the second.
The answer is 258.75

The last time the script was run, the following data was saved:
The first two numbers were the numbers chosen and the last number is the answer to the math operation.
[7, 1, '172.5']

Process finished with exit code 0
```

Figure 9 - Output from PyCharm

Finally, Figure 10 shows the script running in the MacOS Terminal.



```
Assignment07 - zsh - 135x32
Last login: Thu Nov 24 13:39:40 on console
[(base) markbrugger@Marks-MBP ~ % cd /Users/markbrugger/Documents/_PythonClass/Assignment07
(base) markbrugger@Marks-MBP Assignment07 % python3 Assignment07.py
[
*****
readme.txt does not currently exist. It will be created by the program later.
*****

Enter the name of an animal: Cat
Enter the quantity of this animal: 4

*****
Your data has been saved!
*****

['Cat', 4]

Enter a number: 3
Enter a second number: 5

Now we will solve the equation  $23 * X + 11.5 / Y$ , where X is your first number and Y is the second.
The answer is 71.3

The last time the script was run, the following data was saved:
The first two numbers were the numbers chosen and the last number is the answer to the math operation.
[11, 2, '258.75']
(base) markbrugger@Marks-MBP Assignment07 %
```

Figure 10 - MacOS Terminal script output sample

The binary file created by the script is shown in Figure 11. It is not readable by the human eye.

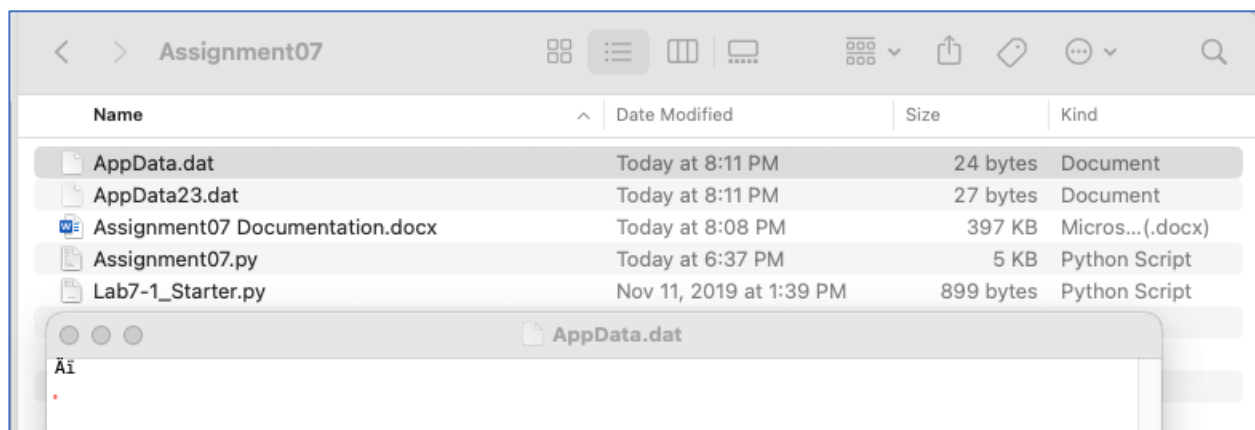


Figure 11 - Output of the script in one of the binary files

Summary

This week we created a demo script from scratch to demonstrate error handling and working with binary files instead of the more typical text file that we had been using. Both concepts were demonstrated separately and then together as one integrated section of the script.