# Natalie E. Tolliver

5/30/2020

ITNFDN 100A

Assignment 07

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

Working with Binary Files & Structured Error Handling

## Introduction

This assignment involves working with GitHub, demonstrating how to work with binary files using Python’s pickling module, and how to work with exception handling. Pickling is a method of reading and writing to and from a binary file using one of Python’s built-in modules. The syntax and the reasoning behind using pickling will be discussed. Following that, structured error handling will be discussed. Structured error handling provides a way for the developer to show user-friendly error messages to the user and makes the developer think of ways in which the user could “break” the program. It can also help guide the troubleshooting process as it could provide information on where the error was encountered by treeing through various levels of structured errors with the general error being placed at the end. Lastly, using GitHub is an integral part of this assignment, so the methodology for GitHub will also be included in this document.

# Learning Objectives

The following learning objectives were useful in understanding this module.

## What is the Markdown language?

Markdown is a easy-to-use syntax for styling all forms of writing on the GitHub platform (<https://guides.github.com/features/mastering-markdown/>, 5.30.2020, External). When you write code in markdown it will get converted into hypertext markup language (html) which is used to create content for the web. In contrast to html, markdown uses regular words (plain text) with added formatting characters so it is easier to read and easier to write than html code. The markdown “parser” converts markdown to html. Files have the extension “.md” for markdown language (<https://www.youtube.com/watch?v=5JwKbeKnhMk>, 5.30.2020, External). It is used for the readme.md file on GitHub and the index.md. An Integrated Development Environment (IDE) can be used to write in the markdown language. Some of the IDEs are Microsoft Visual Studio (<https://visualstudio.microsoft.com/>, 5.30.2020, External), Markdown Pad (<http://markdownpad.com/>, 5.30.2020, External), and Typora (https://typora.io/, 5.30.2020, External). Typora uses the GitHub flavored version of markdown, and live preview. For a more detailed discussion on some of the history of markdown language and markdown editors, this is a useful website: (<https://www.oberlo.com/blog/markdown-editors>, 5.20.2020, External)

## How do you use Markdown on a GitHub webpage?

To use markdown on a GitHub webpage you type in the code using plain text with symbols to indicate the formatting that you would like. For example, # is used for a level 1 heading, ### is used for a level 2 heading, and ### would be used for a level 3 heading with progressively smaller fonts. To make the text italicized, you enclose it in a single asterisk or single underscores. You can use double asterisks or double underlines for strong text. Double tilde encloses text for the strikethrough font. Triple hyphens or triple underscores can be used for a horizontal line. Like Python, if you want to show the actual characters, you can escape them using a \. Git is the actual version control software. GitHub is the website used for version control and collaboration. A great website for explaining GitHub in very non-technical terms is (<https://youtu.be/BCQHnlnPusY>, 5.30.2020, External). He explains using GitHub for version control for a poem, so it was non-technical and very easy to understand the basic concepts. It was awesome for a beginner!

To post a link, you enclose the text you want to display in brackets [], and beside it, the link goes in parentheses. To display an image, the text you want to display is placed in brackets [], the location of the image goes in parentheses. For a code block, you put the ```python on the line before the code and the ``` characters on the line after the code. This website explains a lot of the syntax that will be used for creating the markdown code (<https://www.youtube.com/watch?v=HUBNt18RFbo>, 5.30.2020, External).

# Creating the GitHub WebPage

GitHub is a version control system (VCS) initially created in 2005 for tracking changes to computer code (<https://youtu.be/SWYqp7iY_Tc>, 5.30.2020, External). It can be used for code written in any computer language as it is simply a file storage & version control platform. First you must create a “repository” on Git. A *repository* is like a “project” in Python. A *commit* on GitHub is a record of what files you have changed since the last time you made a commit. When you make changes to your repository (repo), you instruct Git to put those files into your commit history. Commits allow you to go back to the state of a project at any point in time. For this assignment, I started with “Lessons Learned” from the previous assignment. I wrote the markdown code shown in Figure 1 in my docs.index.md file on GitHub (<https://tollivne.github.io/ITFnd100-Mod06/>, 5.30.2020 External)

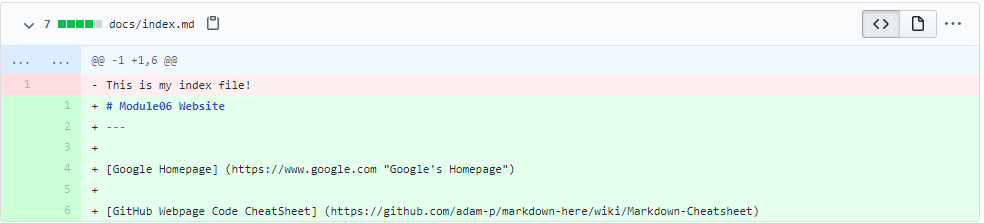


Figure 1 - Markdown Language for Index File

This markdown language created the webpage shown in Figure 6.

A screen shot of a computer

Description automatically generated

Figure 2 - Final Webpage for Assignment 6

This is not how I wanted the webpage to look! So, for Assignment 7, I removed the space between the text I wanted to appear in the link in brackets and the link in parentheses and this solved the problem. Also, of note is the fact that in order to have the 2nd website go to a new line, you must put two spaces after the closing parenthesis. The result is shown in Figure 3.

A screenshot of a cell phone

Description automatically generated

Figure 3 - Resolved Issues from Assignment 6

For assignment 7, this was done in the “readme.md” file. According to the instructions on the GitHub website, the webpage code should reside in the index file. This information is repeated here for convenience in Figure 4.

A screenshot of a cell phone

Description automatically generated

Figure 4 - How to Create a Web Page on GitHub

The webpage markdown code should be placed in the index file because most web servers point to the index file when they receive a request for the directory. This issue is covered eloquently on the Stack Overflow website (<https://stackoverflow.com/questions/35630428/markdown-files-names-is-index-md-mandatory>, 5.30.2020, External). Finally (and this is optional), to choose a theme, click on the “Settings” tab in the repository ribbon, and you can scroll down to pick a theme.

# Naming the Repository

For Assignment 7, there was a requirement to create a repository called **"*IntroToProg-Python-Mod07."*** I had already created a repository called <https://github.com/tollivne/ITFnd100-Mod07>. To learn how to rename the file, the GitHub help website was very helpful (<https://help.github.com/en/github/administering-a-repository/renaming-a-repository>, 5.30.2020, External). To rename it, I went to settings, repositories, and then clicked on the Mod07 repository, I then clicked on Settings in the repository menu, and was able to rename it there. This method is shown in Figure 5.

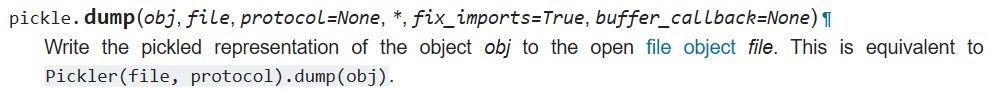
A screenshot of a cell phone

Description automatically generated

Figure 5 - How to Rename a Repository

# Using the Pickling Module in Python

Pickling refers to the process of converting python data into binary. It is also known as “serialization” or “marshalling” or “flattening.” Unpickling refers to the process of converting binary data back into a Python object. To “pickle” or convert a Python object to binary, you call the dump() function. To unpickle the binary data, you call the load() function. An excellent website for explaining and providing the syntax for the pickling module is the official Python software website: (<https://docs.python.org/3/library/pickle.html>, 5.30.2020, External). This is a reliable website that can be trusted to be very accurate and updated for the most recent versions of Python. Although it is very accurate, it can be a bit challenging for a beginner to understand. The pickle dump and load syntax from this website is shown in Figure 6.



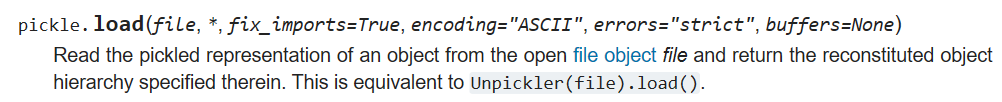


Figure 6 - Pickle Dump & Pickle Load Syntax

The code that I wrote to save the Python object data to a binary file is shown in Figure 7.

def save\_data\_to\_file(file\_name, list\_of\_data):  
 # Now we store the data with the pickle.dump method  
 objFile = open(file\_name, "wb") # write binary  
 pickle.dump(list\_of\_data, objFile)  
 objFile.close

Figure 7 - Saving Python Object to Binary File

The code that I wrote to read the binary data and save it to a Python object is shown in Figure 8.

def read\_data\_from\_file(file\_name):  
 # pass # *TODO: Add code here* # And, we read the data back with the pickle.load method  
 objFile = open("AppData.dat", "rb") # read binary  
 list\_of\_data = pickle.load(objFile) # Load() only loads one row of data  
 objFile.close()  
 return list\_of\_data

Figure 8 - Unpickling - Reading Binary Data

I am a big fan of video tutorials and the following video was great for explaining pickling in Python:

(<https://pythonprogramming.net/python-pickle-module-save-objects-serialization/>, 5.30.2020, External). What I like about this video is that he explained pickling in the beginning much like the official website describes it and then says “Okay, so what does that mean?” Then he explains it in plain English. I was really wondering why you would want to pickle and/or unpickle data in the first place and he explains that too. One of the biggest advantages that I gleaned from the video is that reading in data from a large database can be up to 50 – 100 times faster than just reading the Python data objects. A really good example of the speed of pickled file vs. JSON file can be found on stack overflow: (<https://stackoverflow.com/questions/43056751/why-is-dumping-with-pickle-much-faster-than-json>, 5.30.2020, External). The example uses a timing command to show the time it took to dump to a json vs. a pickled file and shows the results. Stack overflow also contains a really good write-up to help you decide if you should be using pickle (<https://stackoverflow.com/questions/21752259/python-why-pickle>, 5.30.2020, External).

The pickle byte file has the extension “.dat”. Pickle is used only with Python. A note of caution is that pickle is not secure. It is not human readable, and the pickle file could be hacked and contain malicious code. Care should be used if reading someone else’s pickle file or downloading pickle files from the internet. Figure 9 shows a screenshot of the code run using a file called “ToDoFile.txt.”

A screen shot of a computer

Description automatically generated

Figure 9 - Reading & Writing Binary Data

Notice the error message caused intentionally because I tried to read from a file that did not exist. This takes us to the next subject, which is “structured error handling.”

# Structured Error Handling

There are two types of errors in Python. One type is the type of error that occurs when the developer is writing the program (syntax). The other type occurs when the syntax is correct, but the program does not run correctly (run-time) errors. A logic error is one type of run-time error in which the program runs correctly but produces the wrong output, or not what the programmer expected. Another type of runtime error is one that causes the program to crash or stop running. The official Python documentation website is the best source of information on Errors and Exceptions (<https://docs.python.org/3/tutorial/errors.html?highlight=error%20handling>, 5.30.2020, External) but can be very technical. They Python website provides an extensive list of built-in exceptions which is a great reference for looking up exceptions to use in your code (<https://docs.python.org/3/library/exceptions.html>, 5.30.2020, External). The official Python website is a good source to use as a reference if you already understand the basic concepts and just need to look up some syntax or get some clarification.

Error handling is a good way to alert the user to the fact that they may be using the program in a different way than you intended. Instead of giving the user an error written in computereze, you can structure your code so that it gives them an error message that is more sensical. For example, the code in Figure 10 can be used to explain to the user that the file must exist before running the program.

try:  
 objFile = open(file\_name, "rb") # read binary  
  
except FileNotFoundError as e:  
 print("The program is intended to read from a file that already exists!")  
 print("Please make sure the file by the name of :"+ file\_name + " exists!")

Figure 10 - Try-Except for FileNotFound Error

When a call is intentionally made using the wrong file name, a more user-friendly error message is printed out as shown in Figure 11.

A screen shot of a computer

Description automatically generated

The program does not stop running, it continues to the next line in the code which is pickle loading the object file. Since the object file does not exist, and could not be opened, it generates another error, different than the FileNotFoundError. To prevent this, I wanted to exit the program at this point. I imported the system file and used the sys.exit statement to end the program as shown Figure 11 .

import sys

sys.exit()

Figure 11 - Exiting the Program When File Is Not Found

So, even when there was a problem running the program, there are no more difficult to decipher error messages.

The final output has a much nicer appearance as shown in Figure 12.

A picture containing photo, holding, black, white

Description automatically generated

Figure 12 - Final Output of Code with Structured Error Handling

Lastly, a “General” exception error can be placed LAST to be invoked if the error is not caught by one of the earlier more specific errors as shown in Figure 13.

except Exception as e:  
 print("There was a non-specific error!")  
 print("Built-In Python error info: ")  
 print(e, e.\_\_doc\_\_, type(e), sep='\n')

Figure 13 - General Exception Block

Printing the general error and type is added information that can help the developer with troubleshooting. I also added the printing of the error, document string, and type for the FileNotFound error. Shown in Figure 14 is a screenshot of the program running in the CMD window. I was trying to get an error message. The last time I ran a program in the command window, it could not find the file and I had to navigate to the directory containing the python program before running it. I had used a “relative path” in my program. Strangely enough, it didn’t give me the FileNotFoundError this time!

A close up of a screen

Description automatically generated

Figure 14 - Program Run in CMD Window

# Summary

In this assignment, I used Python’s pickling module with the pickle.dump and the pickle.load to write data to a binary file and then read the data and print it out. I also used structured error handling to try to foresee the type of errors a user may get and then print out a message that is more user-friendly than the built-in Python error messages. The assignment involved researching various sources to find more information on pickling and on error handling. I learned that pickling was faster than reading and writing to a JSON file, I learned that Git was created by the same person who created Linux and that it is not a new tool but instead has been around since 2005. I learned a lot about the markdown language in this assignment and was able to post the assignment to the web. I look forward to the next assignment.