Boris Unigovskiy
August 24, 2022
https://github.com/bunigov/IntroToProg-Python-Mod07
https://bunigov.github.io/IntroToProg-Python-Mod07/

IT FDN 100 B
Assignment 7

Working with Pickling and Error Handling

Introduction

In this week's lectures, modules and readings, we continued learning more about how to interact with files (this week's focus was using pickling module to work with binary files) and were introduced to structured error handling. We were provided with a basic understanding but were asked to research pickling and structured error handling on our own, a very important skill to have /have internalized outside the classroom environment. Lastly, we were given some creative freedom and were asked to simply develop an application that will demonstrate the concepts of pickling and structured error handling.

Self-Research on Pickling

I find YouTube to be one of the best sources for understanding technical subjects. Thousands of technical experts give back to the community this way by explaining complex subjects to those entering the field or wanting the refresher. With the abundance of material, if one video doesn't explain clearly, another video likely will. Through these videos, these experts provide interesting commentary and real-life best practices that you likely won't find in technical documentation as well as the ability to interact with the poster and other watchers via comments.

For pickling, I found the following videos to be the most useful. They provided similar information about the *open, dump, load, close* functions but all with slightly different ways of explaining it. Hearing it several times from multiple people helped with the understanding of them content.

https://www.youtube.com/watch?v=2Tw39kZIbhs
https://www.youtube.com/watch?v=6wSFWOleZlc
https://www.youtube.com/watch?v=Pl4Hp8qwwes
https://www.youtube.com/watch?v=qt15PnF8x-M
https://www.youtube.com/watch?v=XzkhtWYYojg
https://www.youtube.com/watch?v=MsRvqMxOryM
https://www.youtube.com/watch?v=YBz3ERXQw_M

Self-Research on Exceptions/Error Handling

For error handling, I took the same approach. Here are the videos that I found the most useful. They also provided similar information about the *try, except, else, finally* structure but with also with different ways of explaining it.

https://www.youtube.com/watch?v=NIWwJbo-9_8
https://www.youtube.com/watch?v=6SPDvPK38tw
https://www.youtube.com/watch?v=nlCKrKGHSSk

Boris Unigovskiy IT FDN 100 B
August 24, 2022 Assignment 7
https://github.com/bunigov/IntroToProg-Python-Mod07
https://bunigov.github.io/IntroToProg-Python-Mod07/

https://www.youtube.com/watch?v=HQqqNBZosn8
https://www.youtube.com/watch?v=MImAiZIzzd4
https://www.youtube.com/watch?v=brICUKrzVR0
https://www.youtube.com/watch?v=KdMAj8Et4xk
https://www.youtube.com/watch?v=65rz-L5ppDo
https://www.youtube.com/watch?v=j_q6NGOwDJo
https://www.youtube.com/watch?v=06HauGzxc9s

Pseudocode for Demonstrating

As I was thinking through how to best demonstrate pickling and error handling, I looked through the last couple of assignments for inspiration. I decided that a combination of storing a list to/restoring from binary (pickle file) and handling exceptions that may arise from interactive with the file would serve as a good demonstration of both concepts.

I decided that my application would do the following:

- Show user list of menu options
- Allow user to see current data in the list (memory)
- Allow user to add data to a list (memory)
- Allow user to remove data from a list (memory)
- Allow user to load data from a pickle file
- Allow user to save data to a pickle file
- Allow user to exit the program

I also decided that my application would perform the following error handling (some of these would not be console error but would simply not make sense so I decided to prevent it as well)

- Handle user attempting to load data from a non-existent file (FileNotFoundError handling)
- Advise user when asking to see current data in the list that the list is empty (custom message)
- Prevent user from saving an empty list to a file (no reason to do this) (custom message)
- Advise user when they've asked for a non-existent Task (in the list) to be removed (custom message)
- Advise user when they've selected an invalid option from the menu (custom message)

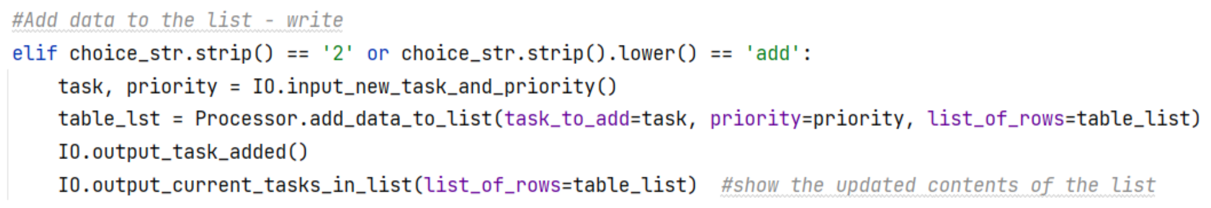
I also decided that I would follow a similar structure of using classes (Processor and IO) and functions for cleaner and more scalable code. The goal was to have processing code in Processor class and input/output code in the IO class as much as possible.

Creating a List

To have some data to work with (and not wanting to hardcode the values), I reused the code from earlier weeks to create a list of dictionaries. This would contain Tasks and Priorities and represent a

ToDo list. The list would be preinitialized by the application with the user having the option to add new tasks to it.

The code to add data to the list looked as shown in **Figure 1** below. It asks the user to first enter a task, then to enter a priority. After, it calls a function to append this data to the list, displays confirmation that it was added and shows the updated contents of the list. I will demonstrate the UI later in this document.



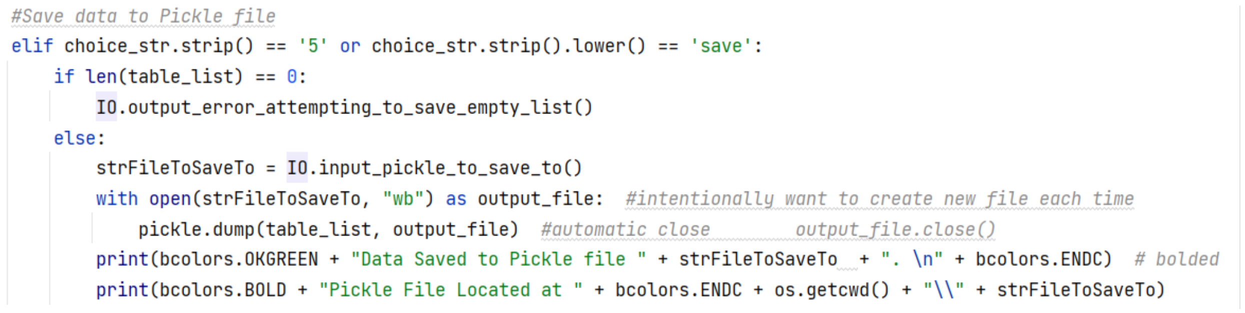
```
#Add data to the list - write
elif choice_str.strip() == '2' or choice_str.strip().lower() == 'add':
    task, priority = IO.input_new_task_and_priority()
    table_lst = Processor.add_data_to_list(task_to_add=task, priority=priority, list_of_rows=table_list)
    IO.output_task_added()
    IO.output_current_tasks_in_list(list_of_rows=table_list)  #show the updated contents of the list
```

Figure 1 – Adding data to the list (code view)

Saving to a Pickle File

Having data in a list (in memory), I was then able to work on the code to load this data to a Pickle file. The code to save the file looked as shown in **Figure 2** below. It first asks the user for a filename to write to (e.g. AppData.dat), opens the file in write binary mode, (intentional decision to overwrite file each time), dumps the list contents to that file and then displays a confirmation message to the user as well as the absolute location of the pickle file (I thought this would be helpful to the user).

Couple things to note here. I decided that it would not make sense to same an empty list to a file so I added an *if* statement that would first check for this logical error and advise the user if the list was empty. Since the list of functions was growing, I intentionally left a couple print statements in this main portion of the code as they were not adding to the complexity.



```
#Save data to Pickle file
elif choice_str.strip() == '5' or choice_str.strip().lower() == 'save':
    if len(table_list) == 0:
        IO.output_error_attempting_to_save_empty_list()
    else:
        strFileToSaveTo = IO.input_pickle_to_save_to()
        with open(strFileToSaveTo, "wb") as output_file:  #intentionally want to create new file each time
            pickle.dump(table_list, output_file)  #automatic close        output_file.close()
        print(bcolors.OKGREEN + "Data Saved to Pickle file " + strFileToSaveTo  + ". \n" + bcolors.ENDC)  # bolded
        print(bcolors.BOLD + "Pickle File Located at " + bcolors.ENDC + os.getcwd() + "\\" + strFileToSaveTo)
```

Figure 2 – Saving list data to Pickle file (code view)
(import pickle statement at the top to enable pickling)

After saving a pickle file, I made sure to validate that the file was there in the directory.

Loading from a Pickle file

While I could see the file was written to the directory, I couldn't validate its contents just yet, so the next step was to write code to load from the Pickle file.

The code to load the file is shown in **Figure 3** below. The code first asks the user for filename of the Pickle file to load from. It attempts to read and load this file into a temporary *objFileData* object in memory (presenting a success message if successful), iterates through that object to get a dictionary row and then builds a *table_list* table object. It completes by presenting confirmation to the user of successful data load into the list and shows the contents of the list (code reuses from previous week's as well as provided as an option if user just wants to the see the contents of the list).

The code uses structured error handling. All of the above commands are in a *try* block so if anything fails (e.g. file is not found), the below *except* block catches the error and presents information to the user gracefully advising them that their file cannot be found (as opposed to the application crashing)

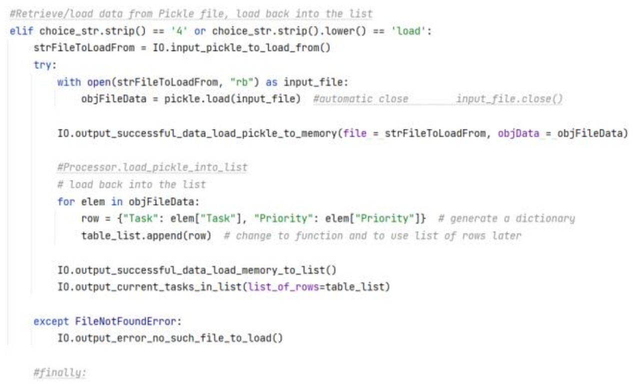

```
#Retrieve/load data from Pickle file, load back into the list
elif choice_str.strip() == '4' or choice_str.strip().lower() == 'load':
    strFileToLoadFrom = IO.input_pickle_to_load_from()
    try:
        with open(strFileToLoadFrom, "rb") as input_file:
            objFileData = pickle.load(input_file)  #automatic close        input_file.close()

        IO.output_successful_data_load_pickle_to_memory(file = strFileToLoadFrom, objData = objFileData)

        #Processor.load_pickle_into_list
        # load back into the list
        for elem in objFileData:
            row = {"Task": elem["Task"], "Priority": elem["Priority"]}  # generate a dictionary
            table_list.append(row)  # change to function and to use list of rows later

        IO.output_successful_data_load_memory_to_list()
        IO.output_current_tasks_in_list(list_of_rows=table_list)

    except FileNotFoundError:
        IO.output_error_no_such_file_to_load()

    #finally:
```

Figure 3 – Loading data from Pickle file (code view)
(import pickle statement at the top to enable pickling)

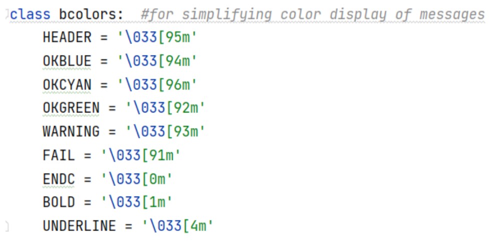
Couple of things to note here. I tested both the happy path (providing the right filename) and unhappy path (providing a non-existent filename). The code handled the happy path and the unhappy path (using the exception block) perfectly. I tested by first creating a list with several tasks, saving to the file, relaunching the application (which starts with empty list), loading the file and validating that the same tasks again appear on the list.

Enhancing the User Experience / Application

The above functionality satisfied the requirements of the assignment to implement pickling (saving and loading) and error handling (handling a *FileNotFoundError*) but I wanted to do more. I had already at every step tried to provide extra messaging / information to the user, but I wanted to add some color to the application.

I did some research and learned about using the ANSI code standard. Specific codes can be inserted into the *print* or *input* command to change text and background colors. Following the following article, I created the same **bcolors** class (**Figure 4** below). I had no need to change it, but it could be customized to any of the ANSI colors, depending on my needs.

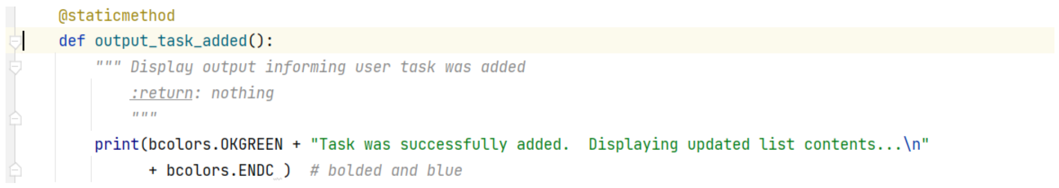
https://stackoverflow.com/questions/287871/how-do-i-print-colored-text-to-the-terminal



```
class bcolors:  #for simplifying color display of messages
    HEADER = '\033[95m'
    OKBLUE = '\033[94m'
    OKCYAN = '\033[96m'
    OKGREEN = '\033[92m'
    WARNING = '\033[93m'
    FAIL = '\033[91m'
    ENDC = '\033[0m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
```

Figure 4 – Custom *bcolors* class for controlling formatting (code view)

Using this custom class, I could then use more recognizable English names to format the output to the user. As shown in **Figure 5** example below, I display a "task successfully added" message in green to the user.



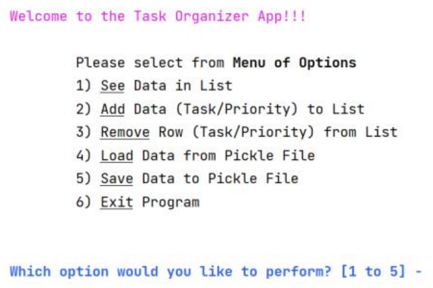
```
    @staticmethod
    def output_task_added():
        """ Display output informing user task was added
            :return: nothing
            """
        print(bcolors.OKGREEN + "Task was successfully added.  Displaying updated list contents...\n"
              + bcolors.ENDC )  # bolded and blue
```

Figure 5 – Using *bcolors* class for controlling formatting when task successfully added (code view)

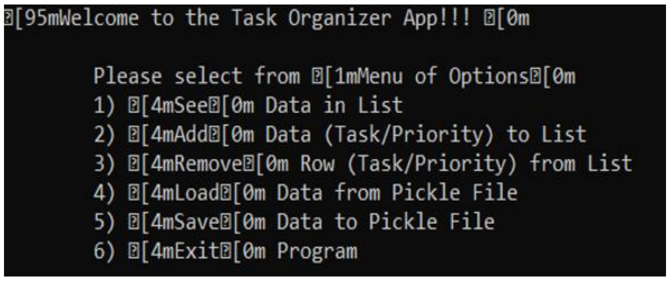
Demo

Here are some of the key screens in the application as demonstrated from PyCharm and Command Prompt.

Figure 6 below shows the welcome screen when the user first launches the application.



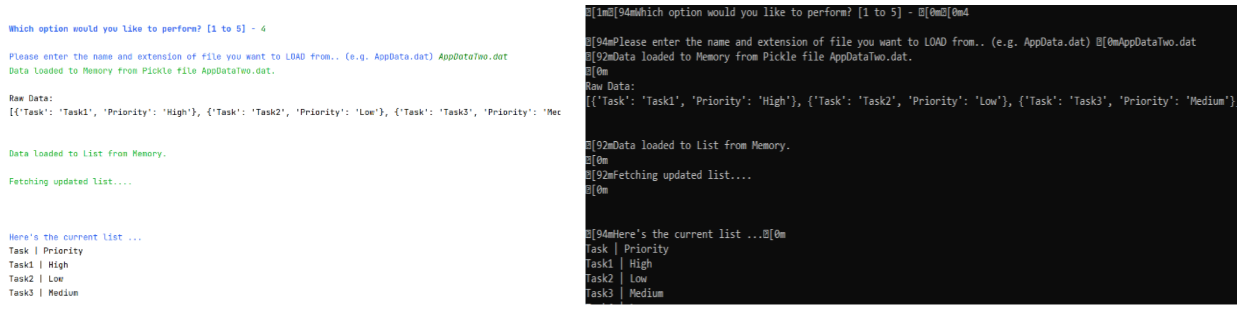
```
Welcome to the Task Organizer App!!!

        Please select from Menu of Options
        1) See Data in List
        2) Add Data (Task/Priority) to List
        3) Remove Row (Task/Priority) from List
        4) Load Data from Pickle File
        5) Save Data to Pickle File
        6) Exit Program


Which option would you like to perform? [1 to 5] -
```



```
[95mWelcome to the Task Organizer App!!! [0m

        Please select from [1mMenu of Options[0m
        1) [4mSee[0m Data in List
        2) [4mAdd[0m Data (Task/Priority) to List
        3) [4mRemove[0m Row (Task/Priority) from List
        4) [4mLoad[0m Data from Pickle File
        5) [4mSave[0m Data to Pickle File
        6) [4mExit[0m Program
```

Figure 6 – Welcome Screens (PyCharm and Command Prompt)

Figure 7 below shows the functionality that allows loading of data from Pickle file.

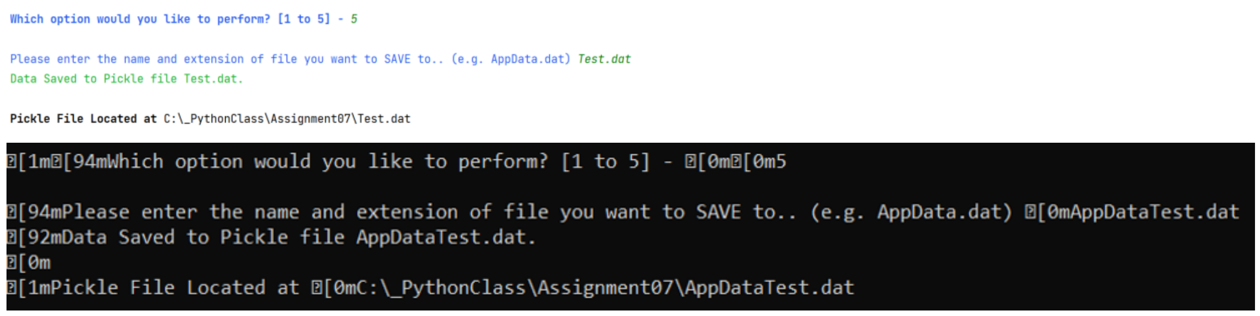


```
Which option would you like to perform? [1 to 5] - 4

Please enter the name and extension of file you want to LOAD from.. (e.g. AppData.dat) AppDataTwo.dat
Data loaded to Memory from Pickle file AppDataTwo.dat.

Raw Data:
[{'Task': 'Task1', 'Priority': 'High'}, {'Task': 'Task2', 'Priority': 'Low'}, {'Task': 'Task3', 'Priority': 'Mec


Data loaded to List from Memory.

Fetching updated list....



Here's the current list ...
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
```

```
[1m[94mWhich option would you like to perform? [1 to 5] - [0m[0m4

[94mPlease enter the name and extension of file you want to LOAD from.. (e.g. AppData.dat) [0mAppDataTwo.dat
[92mData loaded to Memory from Pickle file AppDataTwo.dat.
[0m
Raw Data:
[{'Task': 'Task1', 'Priority': 'High'}, {'Task': 'Task2', 'Priority': 'Low'}, {'Task': 'Task3', 'Priority': 'Medium'}


[92mData loaded to List from Memory.
[0m
[92mFetching updated list....
[0m


[94mHere's the current list ...[0m
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
```

Figure 7 – Loading data from Pickle file (PyCharm and Command Prompt)

Figure 8 below shows the functionality that allows saving data to Pickle file.

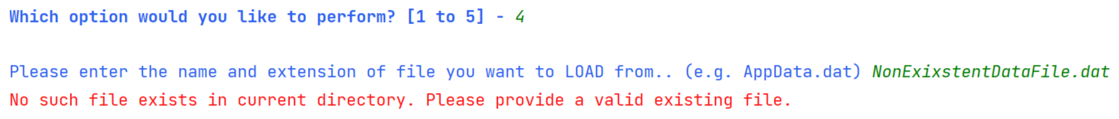


```
Which option would you like to perform? [1 to 5] - 5

Please enter the name and extension of file you want to SAVE to.. (e.g. AppData.dat) Test.dat
Data Saved to Pickle file Test.dat.

Pickle File Located at C:\_PythonClass\Assignment07\Test.dat
```

```
[1m[94mWhich option would you like to perform? [1 to 5] - [0m[0m5

[94mPlease enter the name and extension of file you want to SAVE to.. (e.g. AppData.dat) [0mAppDataTest.dat
[92mData Saved to Pickle file AppDataTest.dat.
[0m
[1mPickle File Located at [0mC:\_PythonClass\Assignment07\AppDataTest.dat
```

Figure 8– Saving Data to Pickle file (PyCharm and Command Prompt)

Figure 9 below shows the error handling functionality when the user attempts to load from a file that does not exist.

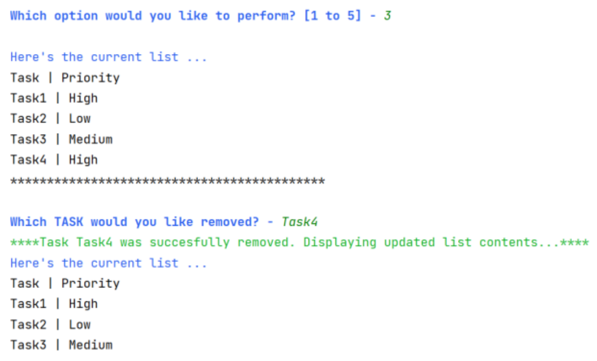


```
Which option would you like to perform? [1 to 5] - 4

Please enter the name and extension of file you want to LOAD from.. (e.g. AppData.dat) NonExixstentDataFile.dat
No such file exists in current directory. Please provide a valid existing file.
```

Figure 9– Attempt to load file that doesn't exist – Error Handling (PyCharm)

Figure 10 below shows the functionality when the user asks to remove a task from the list. This is the happy path when the specified task exists.

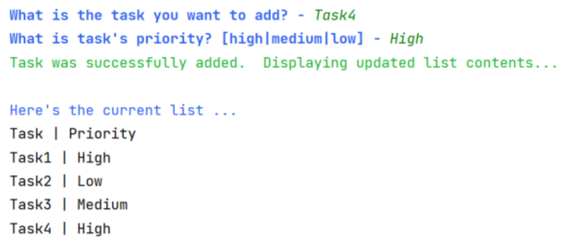
Boris Unigovskiy
August 24, 2022
https://github.com/bunigov/IntroToProg-Python-Mod07
https://bunigov.github.io/IntroToProg-Python-Mod07/



```
Which option would you like to perform? [1 to 5] - 3

Here's the current list ...
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
Task4 | High
*******************************************

Which TASK would you like removed? - Task4
****Task Task4 was succesfully removed. Displaying updated list contents...****
Here's the current list ...
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
```

Figure 10– Removing a Task (PyCharm)

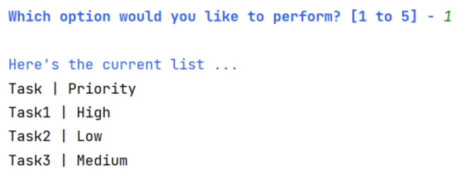
Figure 11 below shows the functionality when the user asks to add a task to the list.



```
What is the task you want to add? - Task4
What is task's priority? [high|medium|low] - High
Task was successfully added.  Displaying updated list contents...

Here's the current list ...
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
Task4 | High
```

Figure 11– Adding a Task (PyCharm)

Figure 12 below shows the functionality when the user asks to see contents of the current list.



```
Which option would you like to perform? [1 to 5] - 1

Here's the current list ...
Task | Priority
Task1 | High
Task2 | Low
Task3 | Medium
```

Figure 12– Seeing Contents of the List (PyCharm)

Summary

This week, we built off our existing knowledge of working with lists/file to now work with binary/pickle files. We also began dealing with exception that may occur from poor user input/ unexpected behavior. We're also continuing to learn about how to build GitHub pages which will serve as a good means to provide a personal summary and a summary of the application to the world.

Looking forward to learning new concepts that will allow me to build more complex and useful applications.