Jordan Keating
May 29, 2023
Foundations of Programming: Python
Assignment 07
https://jskeating.github.io/IntroToProg-Python-Mod07/

Pickling & Error Handling

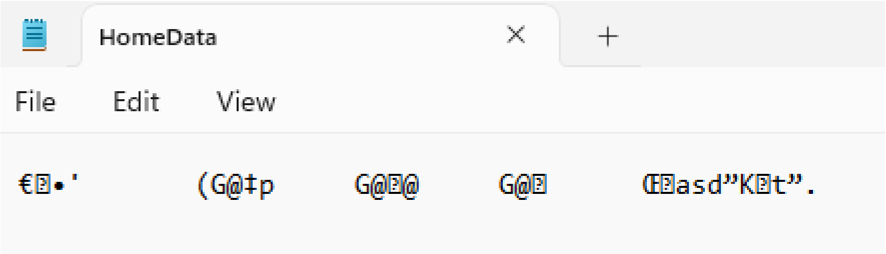
Introduction

This assignment was a bit more open-ended - our task was to create a program that demonstrates both pickling and error handling. I will start out by talking a little bit about the differences between pickled files and the text files we've been working with previously and then move into my idea for a program, a Home Buying/Selling calculator. The program will require quite a bit more time and attention later on to be really show ready for an interview situation, but it was a fun stretch to see what I could come up with while exhibiting these two techniques.

Text files vs. Binary files

So far in this course, we've been reading, writing and appending text files. Binary files work in much the same way, but with a couple of key differences. Binary files save data in bytes while a text file saves data as characters. A text file is easily manipulated from outside of a program (it's easy to read and understand from the NotePad for example), while a binary file would be difficult to change outside of the intended program.

By using pickling, as we'll see later, you can add an additional level of security to the data you intend to store. When opened by a human, these binary files will look almost like complete gibberish. Only when the file is “un-pickled” can one understand what the data means.

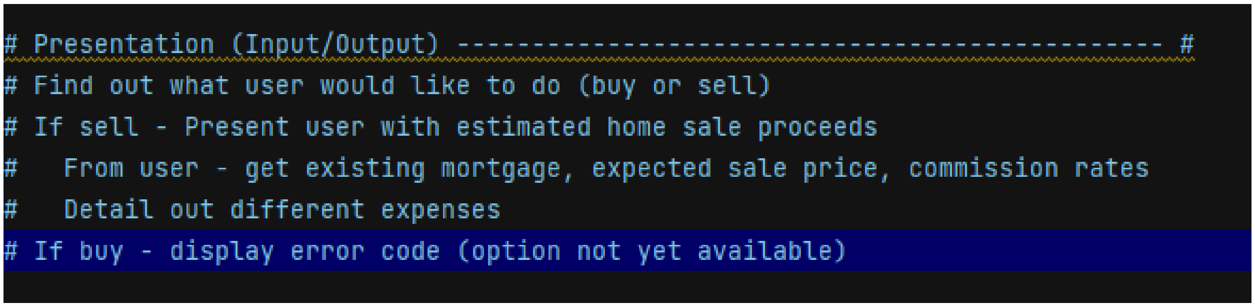


```
€□•'        (G@‡p      G@□@      G@□       Œ□asd”K□t”.
```

Figure 1.1 - A pickled text file

The Home Buy/Sell Calculator

For my script, I decided to demonstrate pickling and error handling with a home-selling based program. I started by using pseudo-code to explain the goals of my program and help me start to outline how I'll accomplish them. I'm going to first ask for user input on whether or not they'd like to buy or sell, then collect some information I would need to help them see their expected proceeds from selling the home (current mortgage, expected sale price, commissions they'll pay to agents, etc). I'll then need to display all of this back to them in an easy to understand block of info. I'm also going to create a custom exception class if the user selects "buy". This will serve two purposes - it allows me to illustrate how to create a custom exception AND will save me some time on this assignment! :)



```
# Presentation (Input/Output) ----------------------------------------------- #
# Find out what user would like to do (buy or sell)
# If sell - Present user with estimated home sale proceeds
#   From user - get existing mortgage, expected sale price, commission rates
#   Detail out different expenses
# If buy - display error code (option not yet available)
```

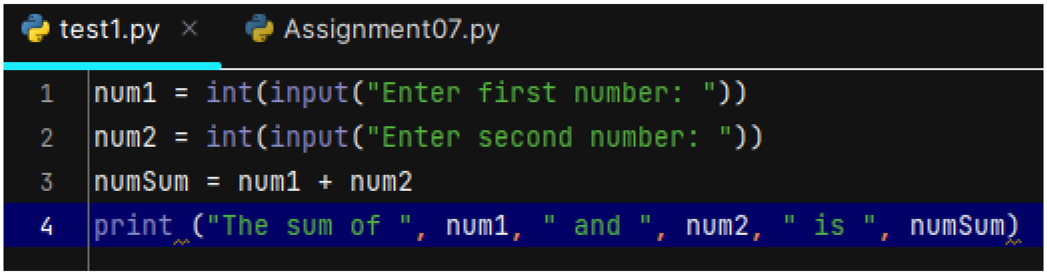
Figure 2.1 - Pseudo-code for Presentation aspects

Our first step is to collect the first user input for their "Buy" or "Sell" choice. I'm going to create an IO (input/output) class with a buy_or_sell function inside of it. This function will collect the user choice and return it to the main body of our script. I'm asking for the user to provide either "B" for Buy or "S" for Sell. This opens us up to a couple of different potential user errors - they could enter a letter we aren't expecting, they could enter a number, they could enter nothing! Let's create some custom errors to catch these different options.

To do this, I'm going to create a Processor function called *choice_check* - *choice_check* is going to take the choice we collected from the *buy_or_sell* input and run it through various checks to ensure we're getting what we want (essentially just "S"). The best way to do this is with a *try/except* block. Before we work on our code, let's back up and show some examples of a *try/except* block in action.

try/except Blocks

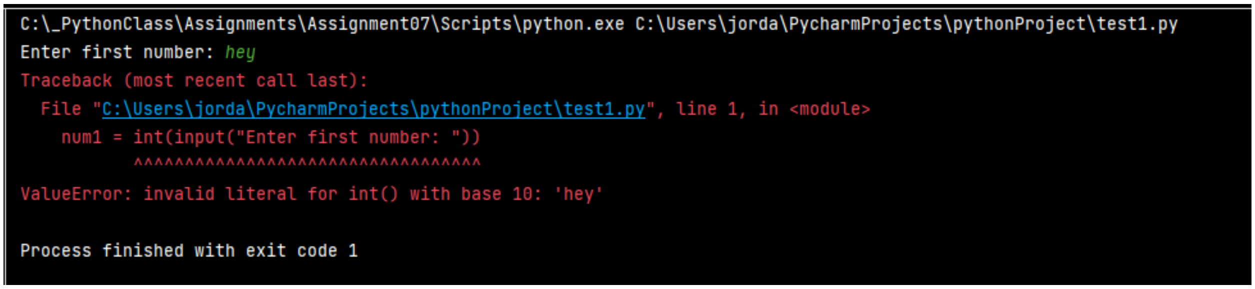
Let's say you are expecting one type of input from a user, but they give you something completely different. Usually, that will end your script with an error message from Python. For example, see Figure 3.1 for a simple addition script.



```python
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
numSum = num1 + num2
print ("The sum of ", num1, " and ", num2, " is ", numSum)
```

Figure 3.1 - A simple addition program adding two user inputs

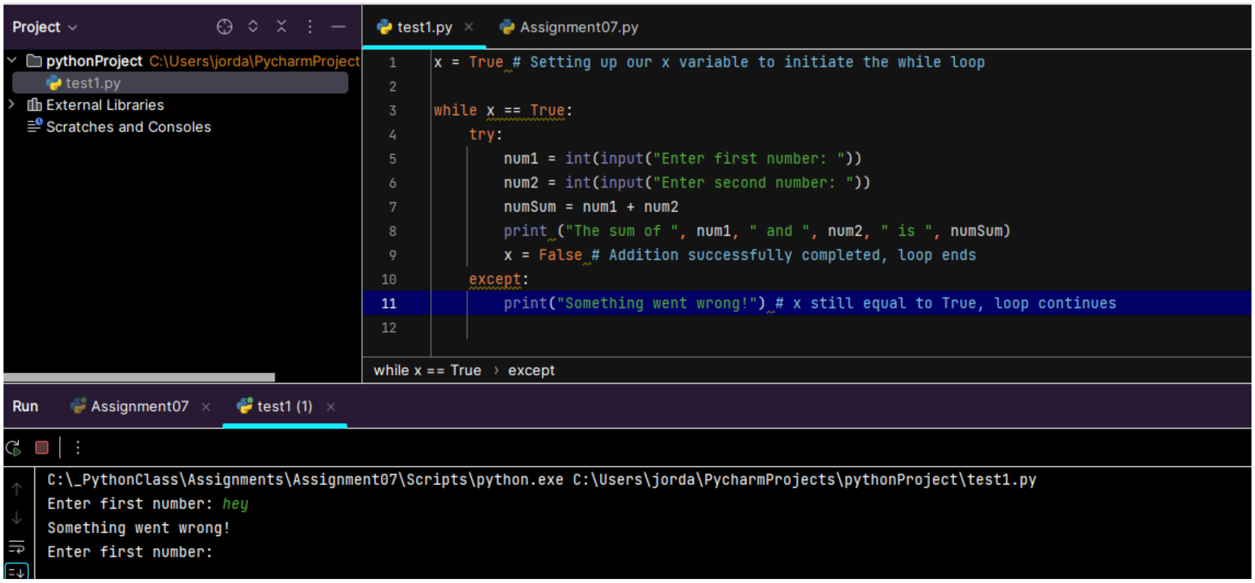
Let’s say a user enters a script variable instead of a number - what happens then? Python will generate an error message and end the program. As you can see in Figure 3.2, the error messages are complicated and somewhat confusing - especially for a user not familiar with the inner workings of your script.



```
C:\_PythonClass\Assignments\Assignment07\Scripts\python.exe C:\Users\jorda\PycharmProjects\pythonProject\test1.py
Enter first number: hey
Traceback (most recent call last):
  File "C:\Users\jorda\PycharmProjects\pythonProject\test1.py", line 1, in <module>
    num1 = int(input("Enter first number: "))
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'hey'

Process finished with exit code 1
```

Figure 3.2 - Python’s ValueError message displayed to the user

By using a *try/except* block, you can accomplish two things - send a more user-friendly error message and keep your program from ending. Let’s set up a *try/except* block within a *while* loop that will keep the program going as long as the user has not entered two integers to be added.



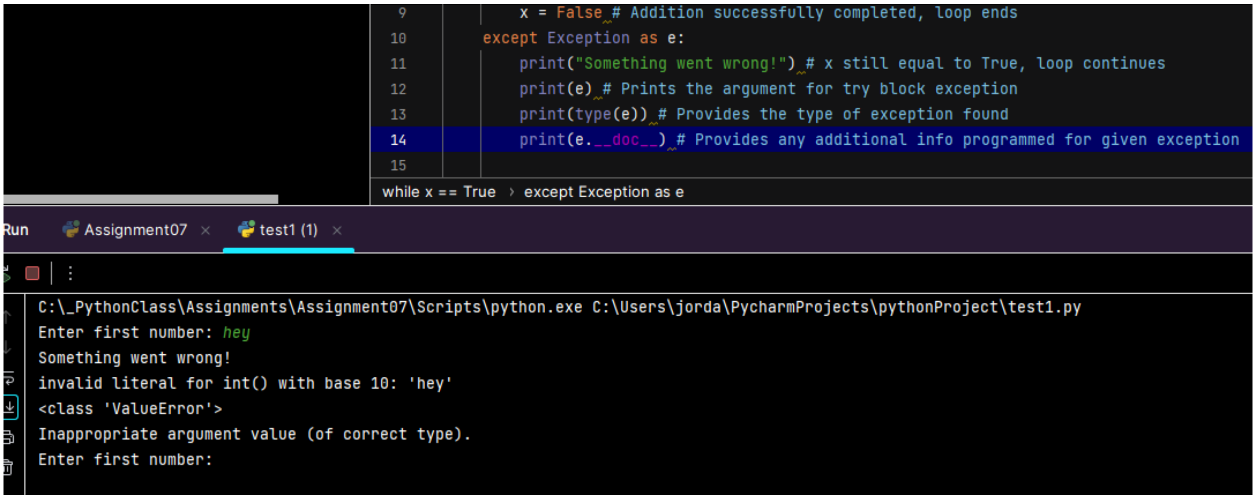
```
x = True # Setting up our x variable to initiate the while loop

while x == True:
    try:
        num1 = int(input("Enter first number: "))
        num2 = int(input("Enter second number: "))
        numSum = num1 + num2
        print ("The sum of ", num1, " and ", num2, " is ", numSum)
        x = False # Addition successfully completed, loop ends
    except:
        print("Something went wrong!") # x still equal to True, loop continues
```

```
C:\_PythonClass\Assignments\Assignment07\Scripts\python.exe C:\Users\jorda\PycharmProjects\pythonProject\test1.py
Enter first number: hey
Something went wrong!
Enter first number:
```

Figure 3.3 - Seeing our *try/except* block in action

The program will first read the *try* section and attempt to run the script there. If an error is encountered, the program jumps to the *except* portion and runs the script there. As you can see in the output, the user entered an invalid option (a string variable where an integer was expected) and the *except* script was run (“Something went wrong!” was printed). Using the *while* loop gives the user another chance to enter the correct variables.

That seems easy enough, but sometimes it may be unclear to you or your user what is actually causing the error. You can use Python’s internal programming to help you identify what the error is and help you add more specificity to your outputs to the user. Using the *as* command, you can capture the argument from the raised exception in your *try* block as a variable and use that to provide the end user more information. See Figure 3.4 for some helpful options.

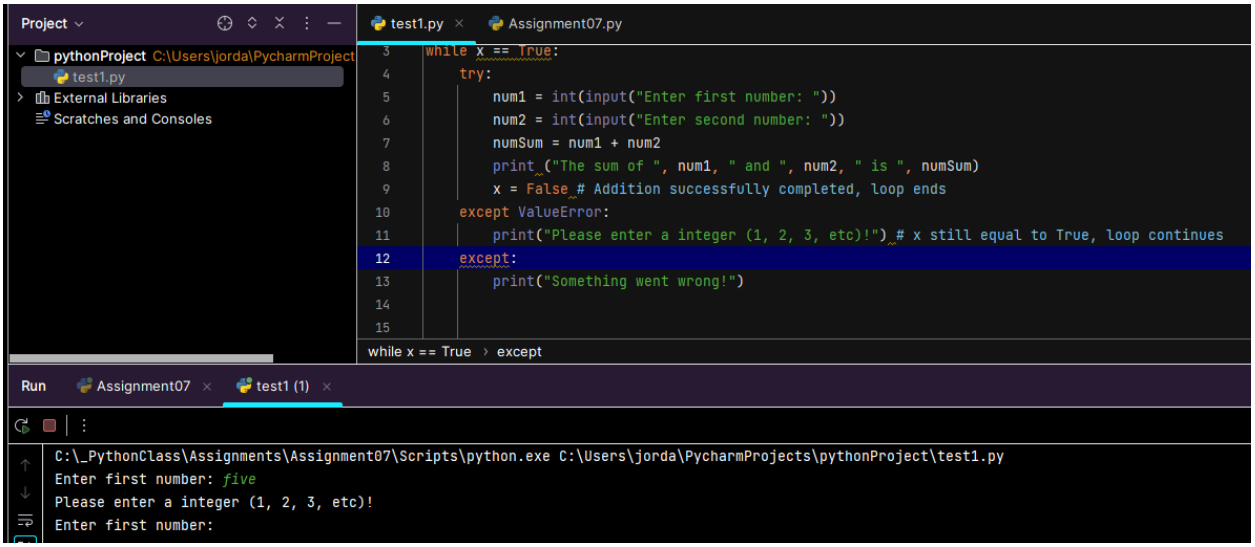


```
        x = False # Addition successfully completed, loop ends
    except Exception as e:
        print("Something went wrong!") # x still equal to True, loop continues
        print(e) # Prints the argument for try block exception
        print(type(e)) # Provides the type of exception found
        print(e.__doc__) # Provides any additional info programmed for given exception
```

```
C:\_PythonClass\Assignments\Assignment07\Scripts\python.exe C:\Users\jorda\PycharmProjects\pythonProject\test1.py
Enter first number: hey
Something went wrong!
invalid literal for int() with base 10: 'hey'
<class 'ValueError'>
Inappropriate argument value (of correct type).
Enter first number:
```

Figure 3.4 - Using the *as* keyword to obtain error message info

By using the *as* keyword and obtaining the error class (ValueError) in this case, we can also set up an exception block specifically to catch that error. This allows us to be more specific in our corrections to the end user. See Figure 3.5 for an updated run.



```
while x == True:
    try:
        num1 = int(input("Enter first number: "))
        num2 = int(input("Enter second number: "))
        numSum = num1 + num2
        print ("The sum of ", num1, " and ", num2, " is ", numSum)
        x = False # Addition successfully completed, loop ends
    except ValueError:
        print("Please enter a integer (1, 2, 3, etc)!") # x still equal to True, loop continues
    except:
        print("Something went wrong!")
```

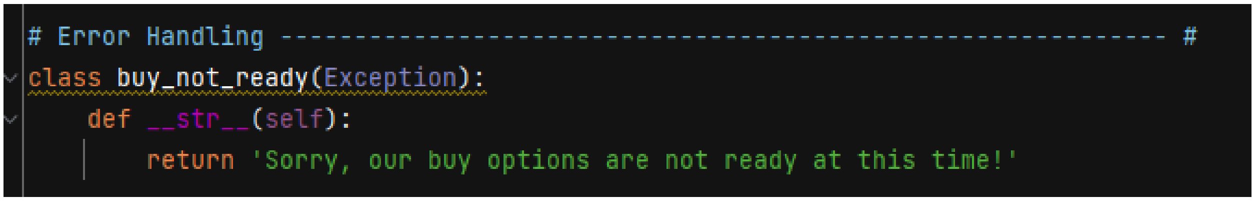
```
C:\_PythonClass\Assignments\Assignment07\Scripts\python.exe C:\Users\jorda\PycharmProjects\pythonProject\test1.py
Enter first number: five
Please enter a integer (1, 2, 3, etc)!
Enter first number:
```

Figure 3.5 - Our error is caught as a *ValueError* and the user is prompted to use an integer

Finally, we can also create our own error catching exceptions and arguments for any custom-errors we'd like within our program. Let's jump back into the main script and see how that works.

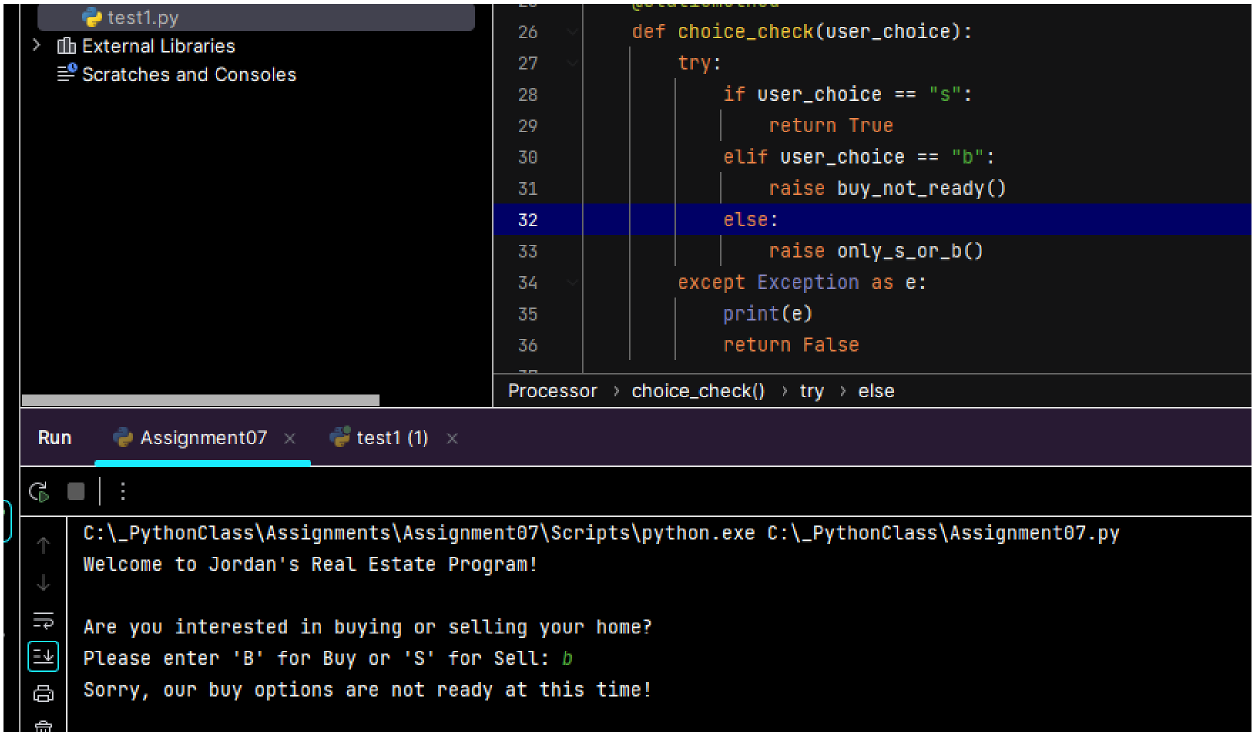
Creating a Custom-Error Block

Each error block is essentially its own class, so we'll create them similarly to how we create our IO and Processing classes. You name the error and define the string to be returned when a user encounters this error. See Figure 4.1 and 4.2 for our first example.



```
# Error Handling ------------------------------------------------------------ #
class buy_not_ready(Exception):
    def __str__(self):
        return 'Sorry, our buy options are not ready at this time!'
```

Figure 4.1 - buy_not_ready error returns message to the except block where the error was raised.

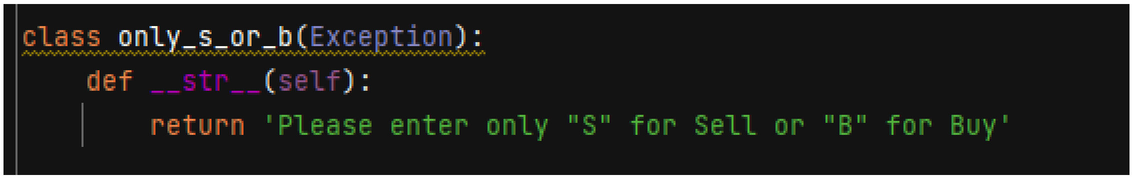


```python
    def choice_check(user_choice):
        try:
            if user_choice == "s":
                return True
            elif user_choice == "b":
                raise buy_not_ready()
            else:
                raise only_s_or_b()
        except Exception as e:
            print(e)
            return False
```

```
C:\_PythonClass\Assignments\Assignment07\Scripts\python.exe C:\_PythonClass\Assignment07.py
Welcome to Jordan's Real Estate Program!

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: b
Sorry, our buy options are not ready at this time!
```

Figure 4.2 - User enters “b” and triggers our custom buy_not_ready exception

We’ll also create another block in case the user enters anything else other than the “s” that we want.



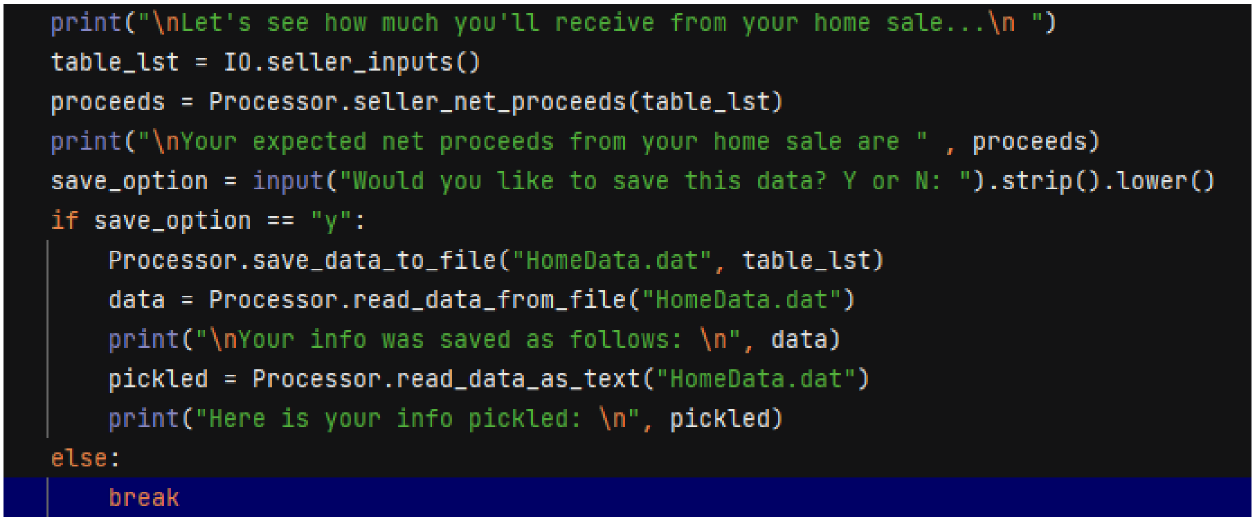
```python
class only_s_or_b(Exception):
    def __str__(self):
        return 'Please enter only "S" for Sell or "B" for Buy'
```

Figure 4.3 - User errors with input other than “s” or “b” and is reminded of available inputs

With the special errors dealt with, I’m going to skip ahead through some of my basic coding and move to the pickling section of the program.

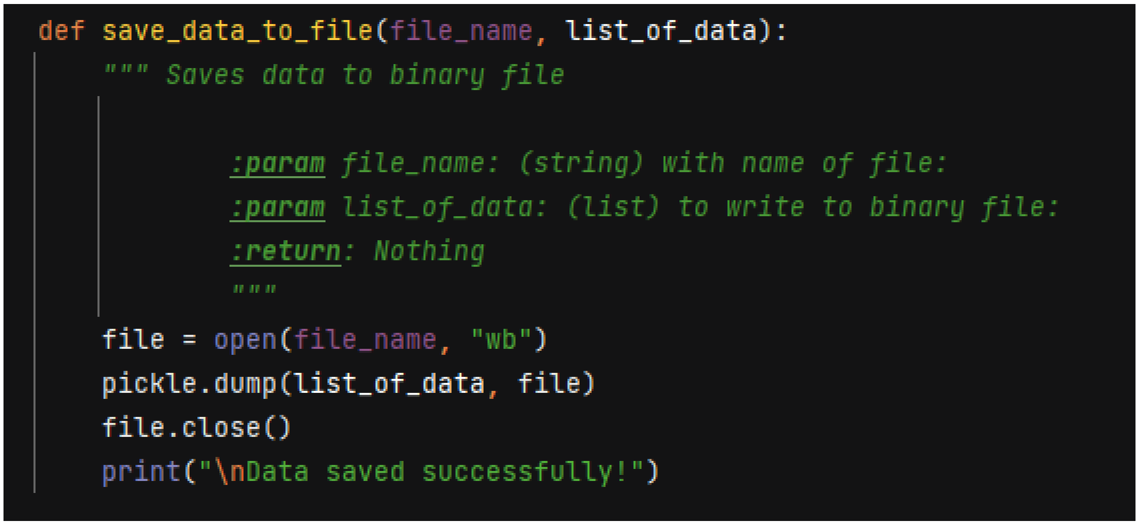
Pickling your data

As we discussed above, pickling your data into a binary file can be extremely useful. While I have big dreams for my Home Buy/Sell program, for now I’ll simply settle with saving the data we acquired from the user into a binary file called **HomeData.dat**. I collected information from the user through an IO function titled *seller_inputs* and used those to make some calculations about their potential proceeds from selling their home. I’m going to also save that data for potential future use. I'll prompt the user to save and if yes, we'll run our *save_data_to_file* processor function. To use the pickle functions, you must first use the *import pickle* command to load these functions for use - we'll be using both *pickle.load()* and *pickle.dump()*. See Figures 5.1 through 5.3 to see them in action



```
print("\nLet's see how much you'll receive from your home sale...\n ")
table_lst = IO.seller_inputs()
proceeds = Processor.seller_net_proceeds(table_lst)
print("\nYour expected net proceeds from your home sale are " , proceeds)
save_option = input("Would you like to save this data? Y or N: ").strip().lower()
if save_option == "y":
    Processor.save_data_to_file("HomeData.dat", table_lst)
    data = Processor.read_data_from_file("HomeData.dat")
    print("\nYour info was saved as follows: \n", data)
    pickled = Processor.read_data_as_text("HomeData.dat")
    print("Here is your info pickled: \n", pickled)
else:
    break
```

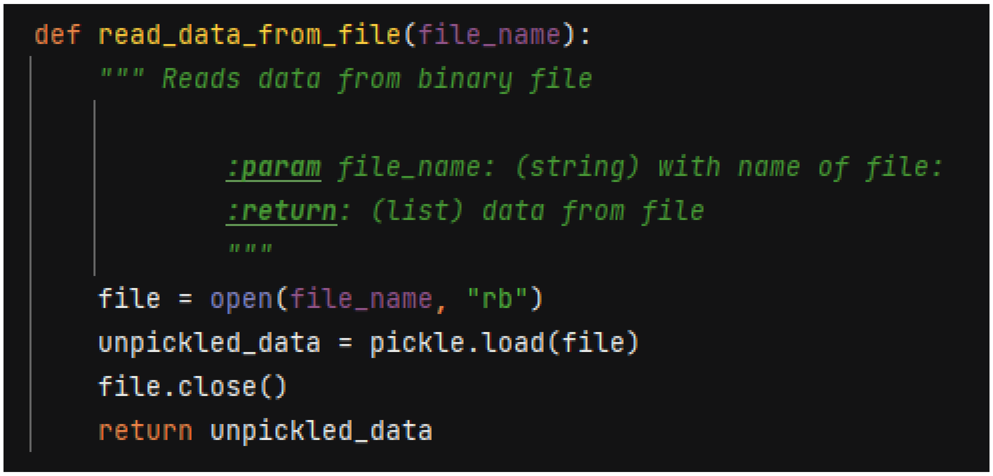
Figure 5.1 - Main body of script



```
def save_data_to_file(file_name, list_of_data):
    """ Saves data to binary file

        :param file_name: (string) with name of file:
        :param list_of_data: (list) to write to binary file:
        :return: Nothing
        """
    file = open(file_name, "wb")
    pickle.dump(list_of_data, file)
    file.close()
    print("\nData saved successfully!")
```

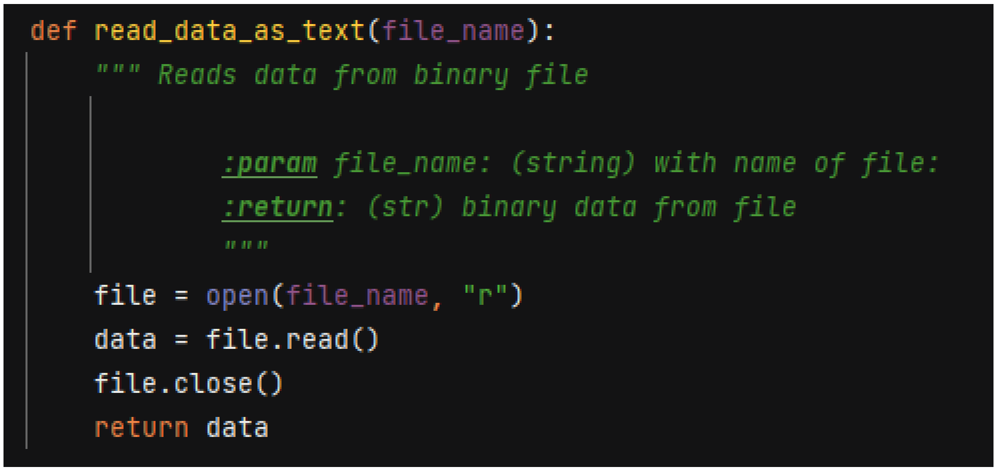
Figure 5.2 - *save_data_to_file* processor function

To ensure the data was saved as a binary file, I'll run the program below and check our outputs. I'm also opening our binary file and reading it back (see *read_data_from_file* function in Figure 5.3) and then also opening the saved binary file as if it were a text file to see the "pickled" data within our program (see Figure 5.4).



```python
def read_data_from_file(file_name):
    """ Reads data from binary file

            :param file_name: (string) with name of file:
            :return: (list) data from file
            """
    file = open(file_name, "rb")
    unpickled_data = pickle.load(file)
    file.close()
    return unpickled_data
```

Figure 5.3 - Reading our binary file

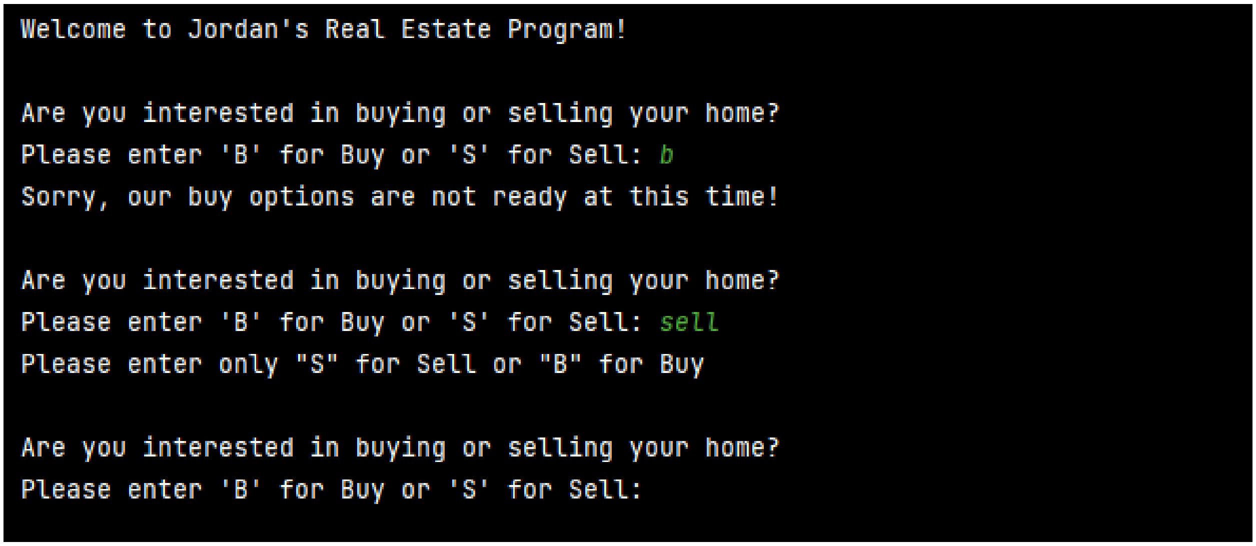


```python
def read_data_as_text(file_name):
    """ Reads data from binary file

            :param file_name: (string) with name of file:
            :return: (str) binary data from file
            """
    file = open(file_name, "r")
    data = file.read()
    file.close()
    return data
```

Figure 5.4 - Reading our binary file as a text file

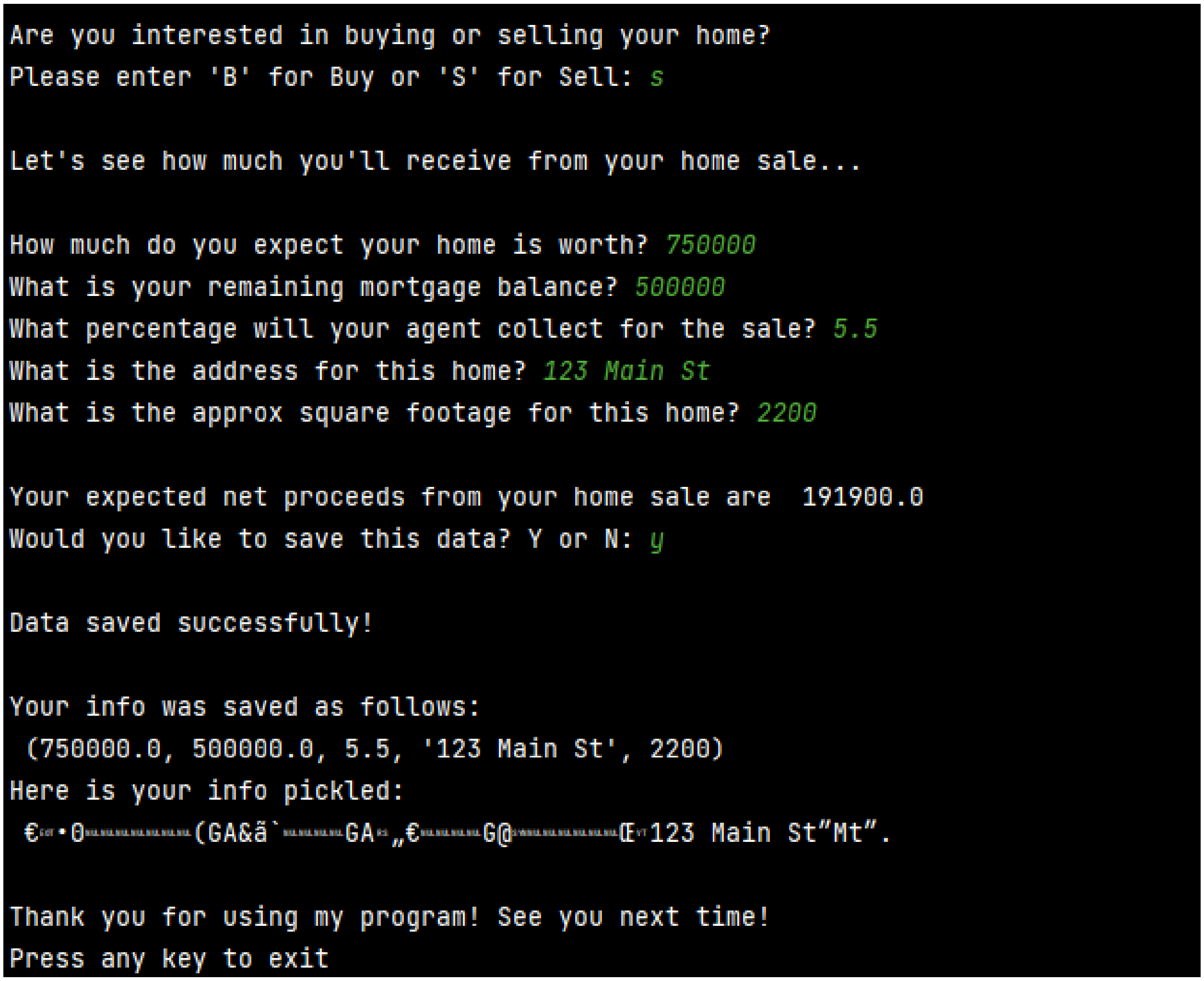
Testing

Alright, we'll now test all features in both PyCharm and CMD to ensure our script is working as intended.



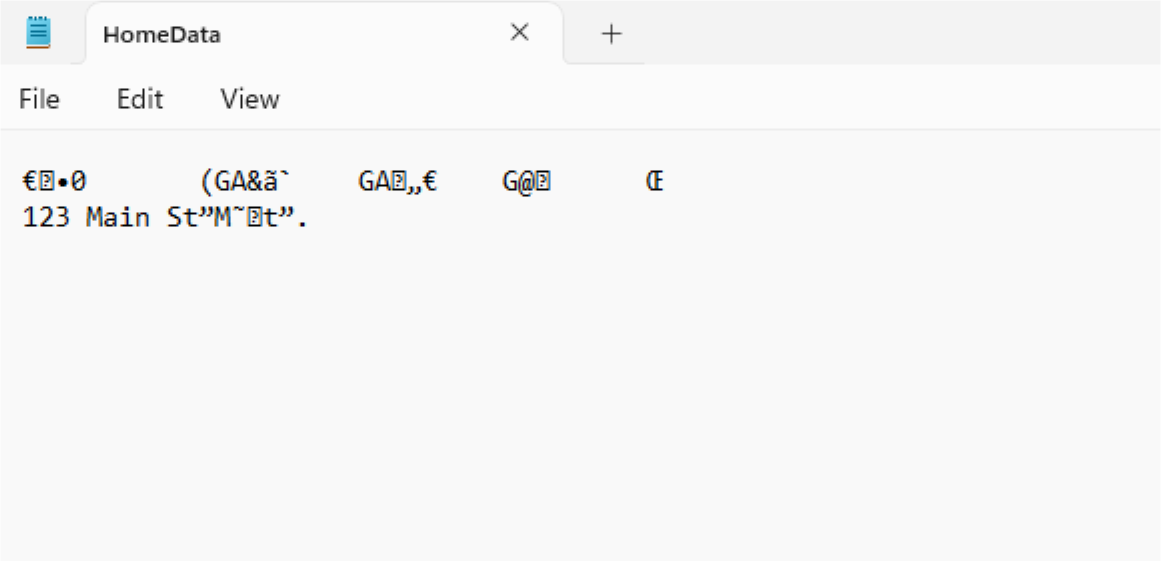
```
Welcome to Jordan's Real Estate Program!

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: b
Sorry, our buy options are not ready at this time!

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: sell
Please enter only "S" for Sell or "B" for Buy

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell:
```

Figure 6.1 - Running in PyCharm - Catching both custom errors successfully and looping back into program



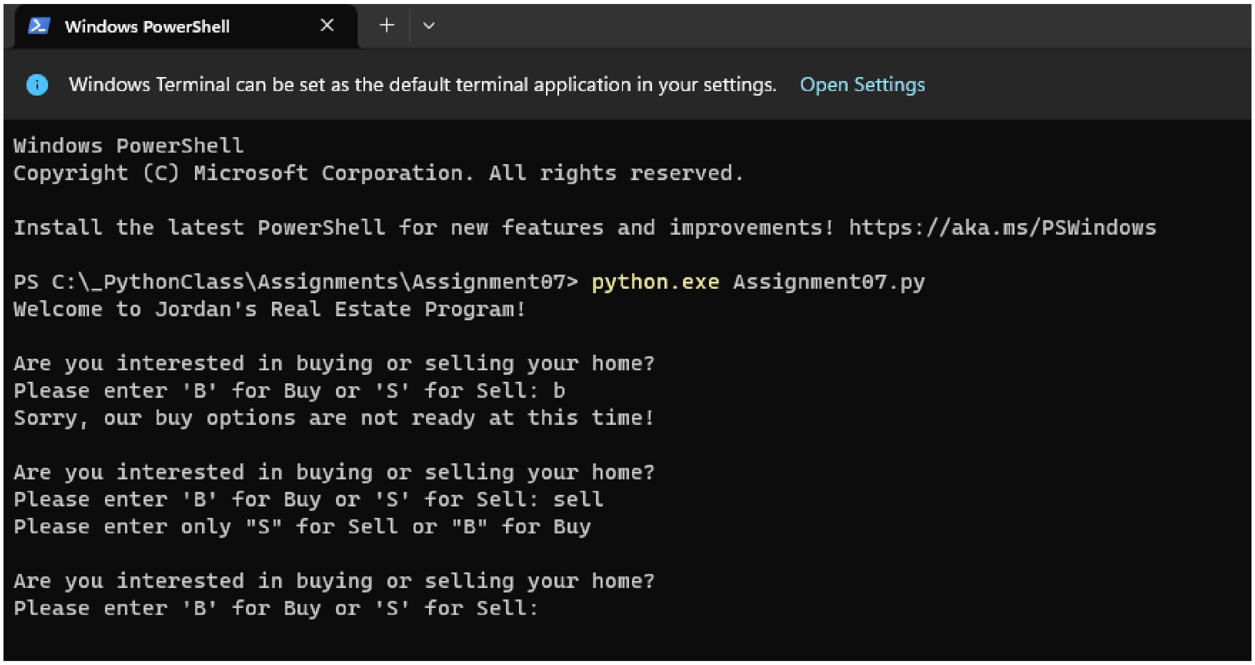
```
Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: s

Let's see how much you'll receive from your home sale...

How much do you expect your home is worth? 750000
What is your remaining mortgage balance? 500000
What percentage will your agent collect for the sale? 5.5
What is the address for this home? 123 Main St
What is the approx square footage for this home? 2200

Your expected net proceeds from your home sale are  191900.0
Would you like to save this data? Y or N: y

Data saved successfully!

Your info was saved as follows:
 (750000.0, 500000.0, 5.5, '123 Main St', 2200)
Here is your info pickled:
 €•0(GA&ã`GA„€G@Œ123 Main St”Mt”.

Thank you for using my program! See you next time!
Press any key to exit
```

Figure 6.2 - Running in PyCharm - “Sell” option collects user data, calculates proceeds, and saves into pickled binary file which is then read back and displayed to user (both pickled and unpickled) before exiting the program



```
€•0        (GA&ã`     GA„€     G@       Œ
123 Main St”M˜t”.
```

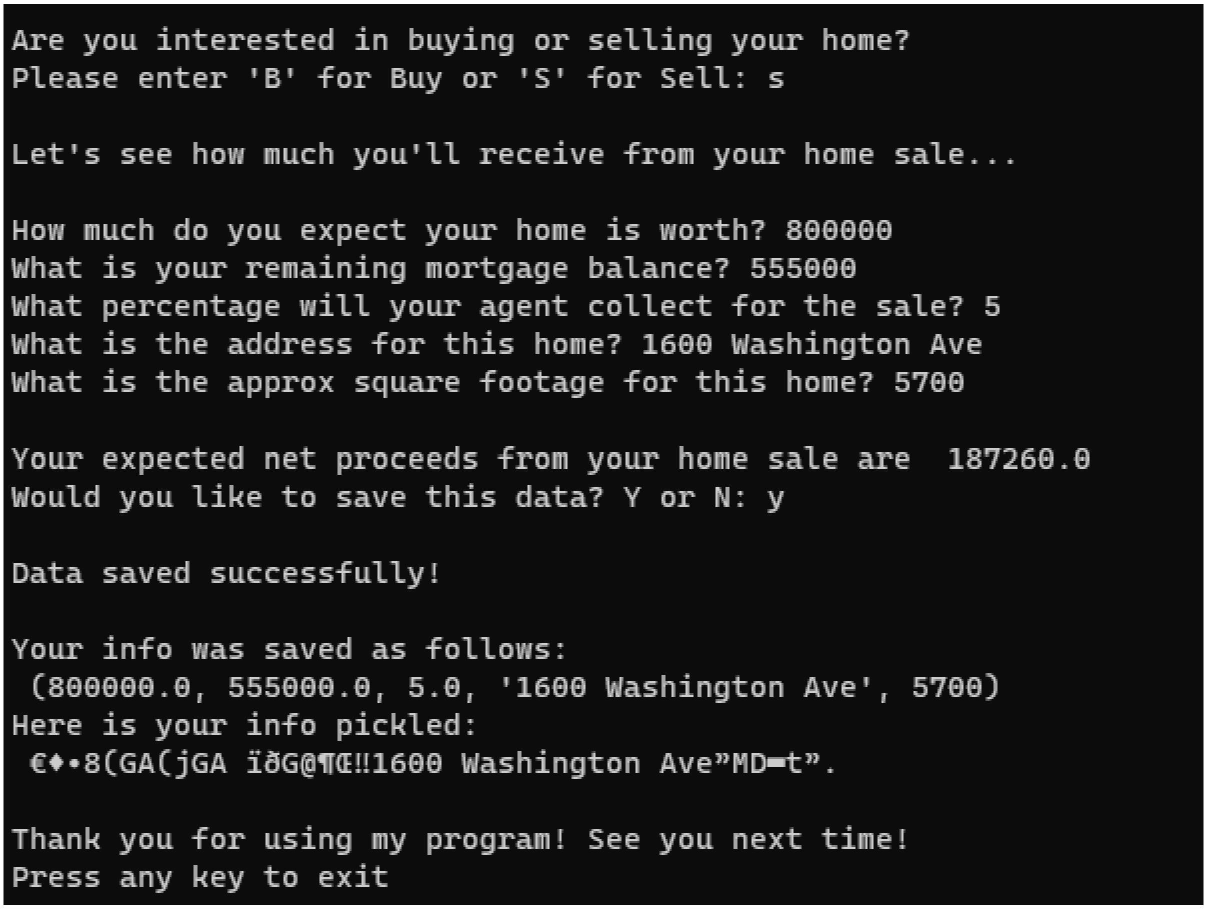
Figure 6.3 - HomeData.dat file updated successfully - “123 Main St” is visible from user inputs



Windows Terminal can be set as the default terminal application in your settings.

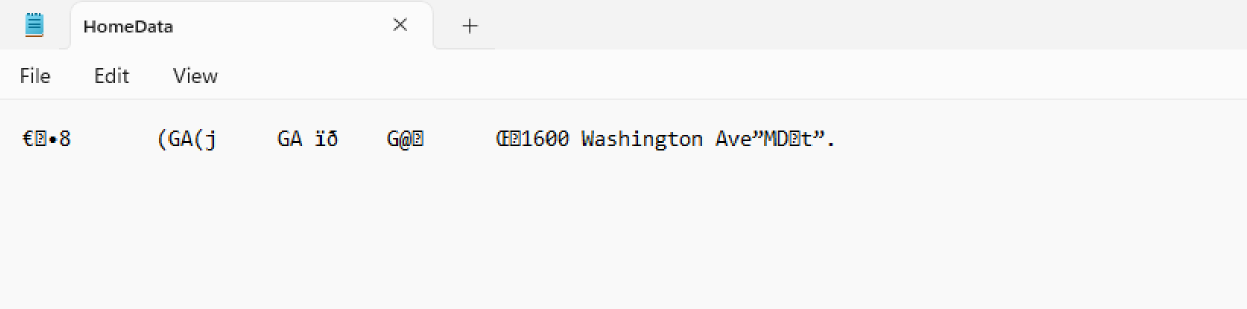
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\_PythonClass\Assignments\Assignment07> python.exe Assignment07.py
Welcome to Jordan's Real Estate Program!

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: b
Sorry, our buy options are not ready at this time!

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: sell
Please enter only "S" for Sell or "B" for Buy

Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell:
```

Figure 6.4 - Running in CMD - Catching both custom errors successfully and looping back into program



```
Are you interested in buying or selling your home?
Please enter 'B' for Buy or 'S' for Sell: s

Let's see how much you'll receive from your home sale...

How much do you expect your home is worth? 800000
What is your remaining mortgage balance? 555000
What percentage will your agent collect for the sale? 5
What is the address for this home? 1600 Washington Ave
What is the approx square footage for this home? 5700

Your expected net proceeds from your home sale are  187260.0
Would you like to save this data? Y or N: y

Data saved successfully!

Your info was saved as follows:
 (800000.0, 555000.0, 5.0, '1600 Washington Ave', 5700)
Here is your info pickled:
 €♦•8(GA(jGA ïðG@¶Œ‼1600 Washington Ave”MD▬t”.

Thank you for using my program! See you next time!
Press any key to exit
```

Figure 6.5 - Running in CMD - “Sell” option collects user data, calculates proceeds, and saves into pickled binary file which is then read back and displayed to user (both pickled and unpickled) before exiting the program



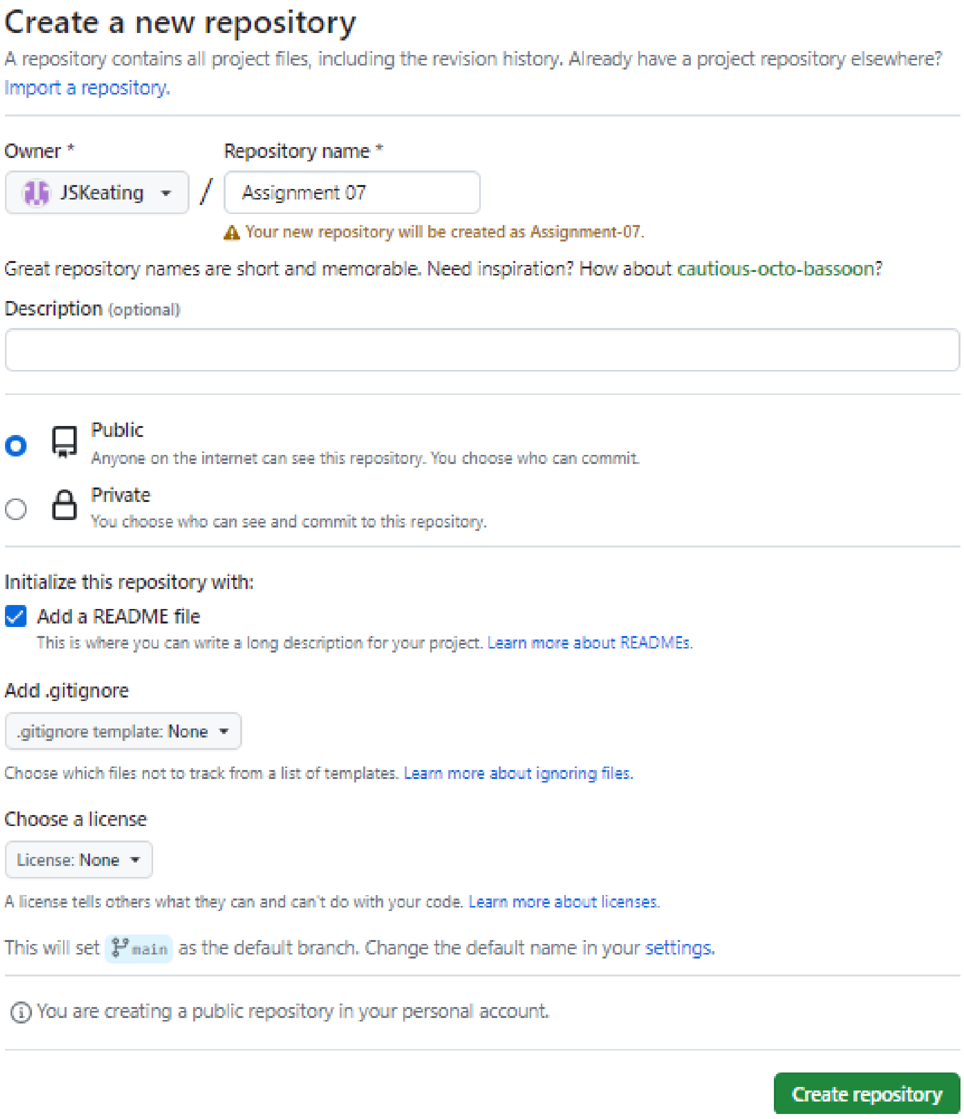
```
€▯•8        (GA(j      GA ïð     G@▯       Œ▯1600 Washington Ave”MD▯t”.
```

Figure 6.6 - Confirming HomeData.dat changes - “1600 Washington Ave” visible

Building a Git-Hub website

As the last piece of this assignment, we'll be re-building this doc into a GitHub website. This is an awesome skill to have in your toolbelt - these can be super handy for future job interviews, reference for other projects, or just showing off your work to friends and family.

We start by creating a new repository for this Module (see Figure 7.1) and then creating a new index.md file which is where we'll do our actual editing (see Figure 7.2). To create the new repository, log into your GitHub account and select the green "New" button on your Repositories tab. Remember to add a ReadMe.md file to the repository for any comments you'll want to make for your viewers.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * JSKeating / Repository name * Assignment 07

⚠ Your new repository will be created as Assignment-07.

Great repository names are short and memorable. Need inspiration? How about cautious-octo-bassoon?

Description (optional)

● Public
Anyone on the internet can see this repository. You choose who can commit.

○ Private
You choose who can see and commit to this repository.

Initialize this repository with:

☑ Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore

.gitignore template: None

Choose which files not to track from a list of templates. Learn more about ignoring files.

Choose a license

License: None

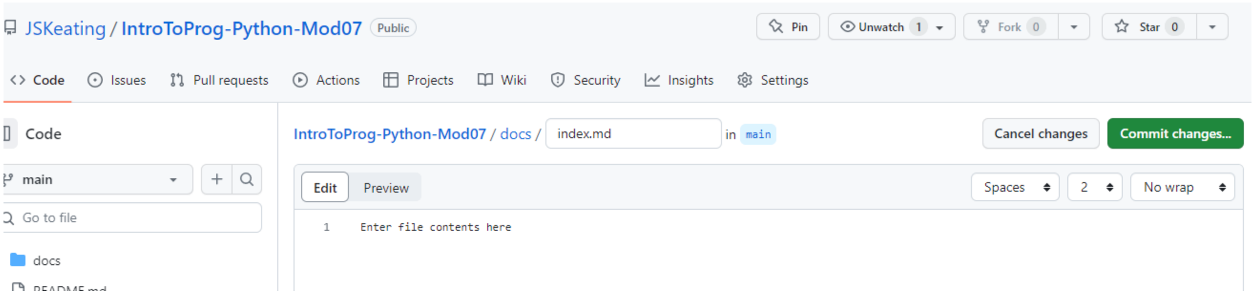
A license tells others what they can and can't do with your code. Learn more about licenses.

This will set main as the default branch. Change the default name in your settings.

ⓘ You are creating a public repository in your personal account.

Figure 7.1 - Creating a new repository

To add the index.md page, click on the “Add New File” button on your repository page. At the top of the page, you’ll see a file path prompting you to “Name your file…”. In this box, you’ll want to type “docs” followed by a forward slash and then “index.md”. This will create a “docs” folder and place your index.md file inside it.



IntroToProg-Python-Mod07 / docs / index.md in main

Figure 7.2 - Creating the index.md file

The index.md file is where we’ll work on recreating our document. Click on the pencil tool to begin editing and you’ll see you have two panes to view your file in - “Edit” and “Preview”. Edit allows you to use “markdown” language to format your text. Markdown has many different uses, but for our purposes in this assignment, I’m mostly going to be using the single and double asterisks (single for italics, double for bold), the hashes (single hash is large header, double hash is slightly smaller, and so on up to six hashes), and coding to insert photos. Much more info can be found in this GitHub article: Basic writing and formatting syntax - GitHub Docs.

Summary

In this assignment, we set off on our own adventure in creating a program. I decided to begin creating a real estate calculator that I can hopefully one day share with my clients (as a Real Estate agent). The calculator helped me to demonstrate error handling as it occurs in Python, how we can create custom error blocks, and how we can use *try/except* blocks to prevent these errors from stopping our program in its tracks. We also looked at how this data can be stored more efficiently and securely by importing the pickle class and utilizing the functions there to create and read binary files.