Matthew Tsao

November 29, 2022

Foundations Of Programming: Python

Assignment07

GitHub URL: https://github.com/mrtsao1/IntroToProg-Python-Mod07

GibHub Webpage: https://github.com/mrtsao1/IntroToProg-Python-Mod07/blob/main/docs/index.md

# Create Script using Pickling and Error Handling

## Introduction

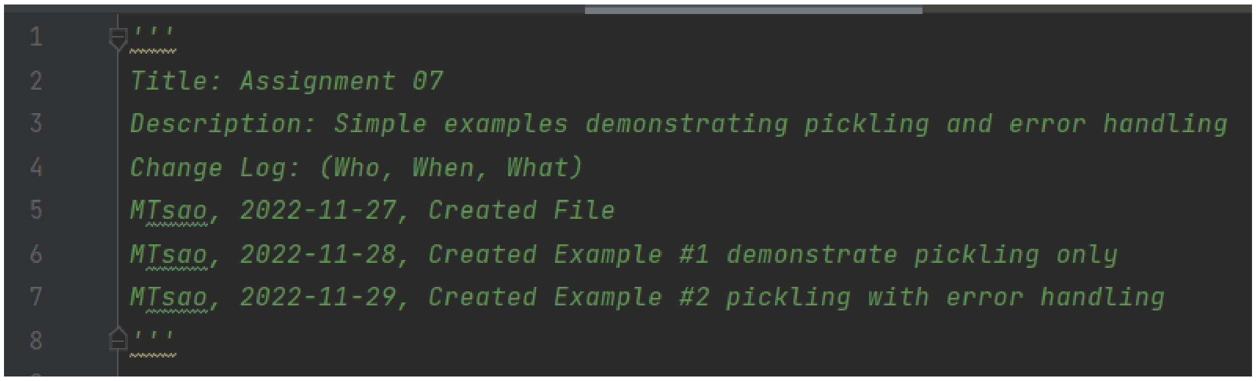
This week, I learned about pickling and error handling. I created a script to demonstrate these topics. The script consists of two parts: ***Example #1 – Pickling*** and ***Example #2 – Error Handling and Raising Custom Errors***. The following information is a breakdown on how I wrote this program step-by-step.

## Creating and Running the Program

Refer to the following subsections below.

### Script Header

Create the script header. A script header should contain the title of the script file, a description, and a change log. A script header is created using comments, which consists of either starting the line with "#" (single line comment) or multi-line comments (Figure 1).

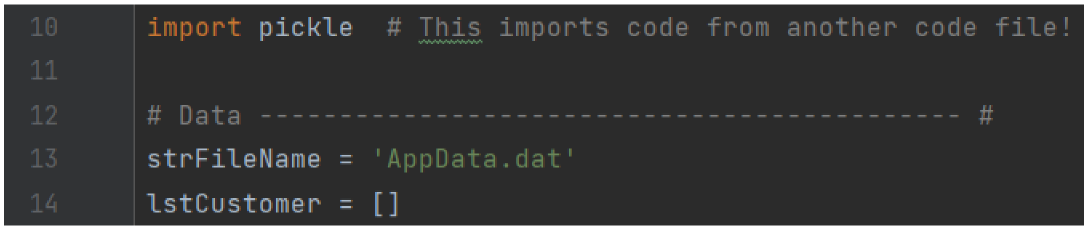


```
'''
Title: Assignment 07
Description: Simple examples demonstrating pickling and error handling
Change Log: (Who, When, What)
MTsao, 2022-11-27, Created File
MTsao, 2022-11-28, Created Example #1 demonstrate pickling only
MTsao, 2022-11-29, Created Example #2 pickling with error handling
'''
```

*Figure 1. Script Header*

## Import Pickle Module and Initialize Variables

Import pickle module and create variables with initialized values that will be used throughout the program (Figure 2). The key variables and their descriptions are shown in figure 2 below. The pickle module will be used to work with binary files.



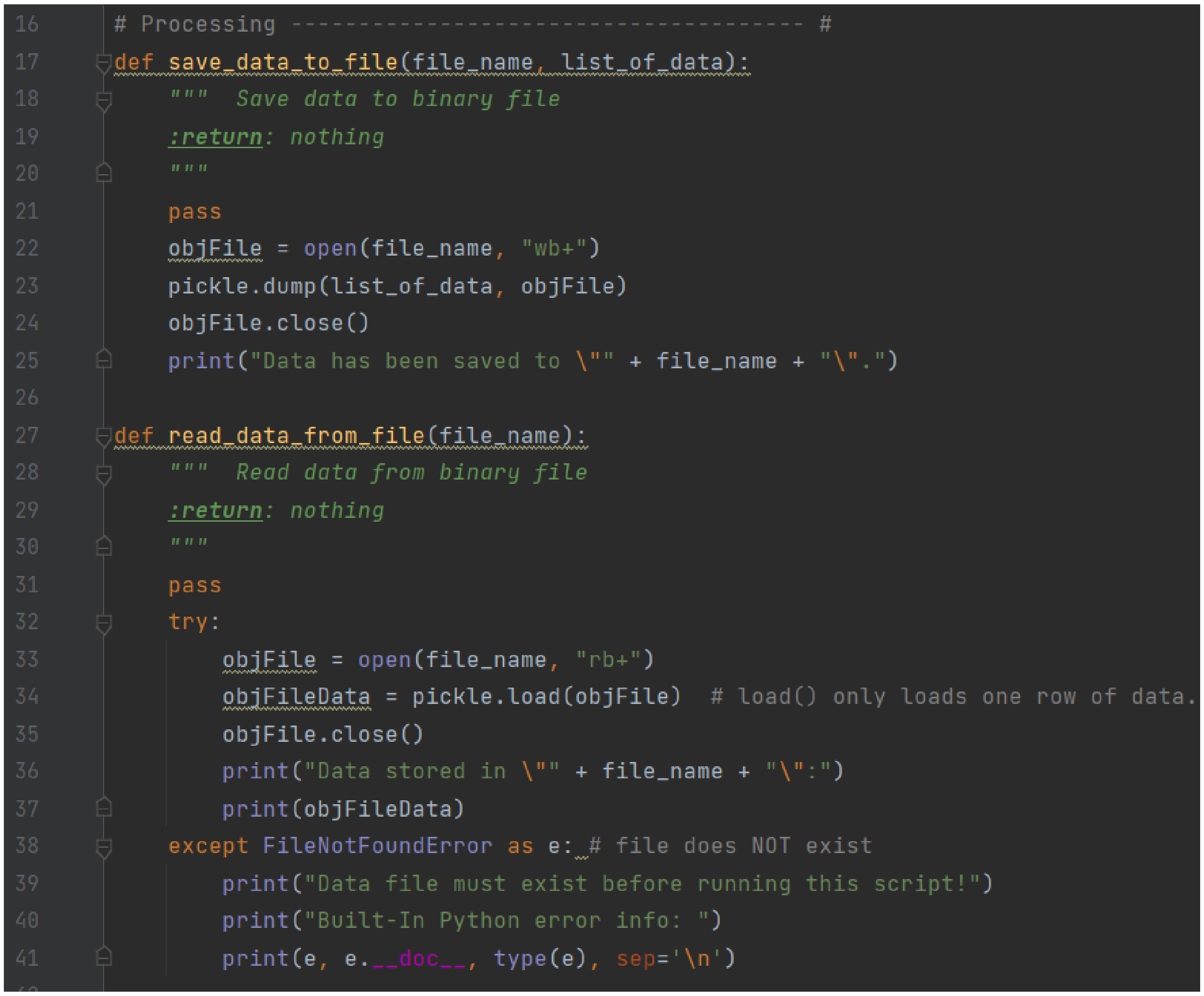
```python
import pickle  # This imports code from another code file!

# Data ------------------------------------------- #
strFileName = 'AppData.dat'
lstCustomer = []
```

*Figure 2. Import Pickle Module and Initialize Variables*

## Functions

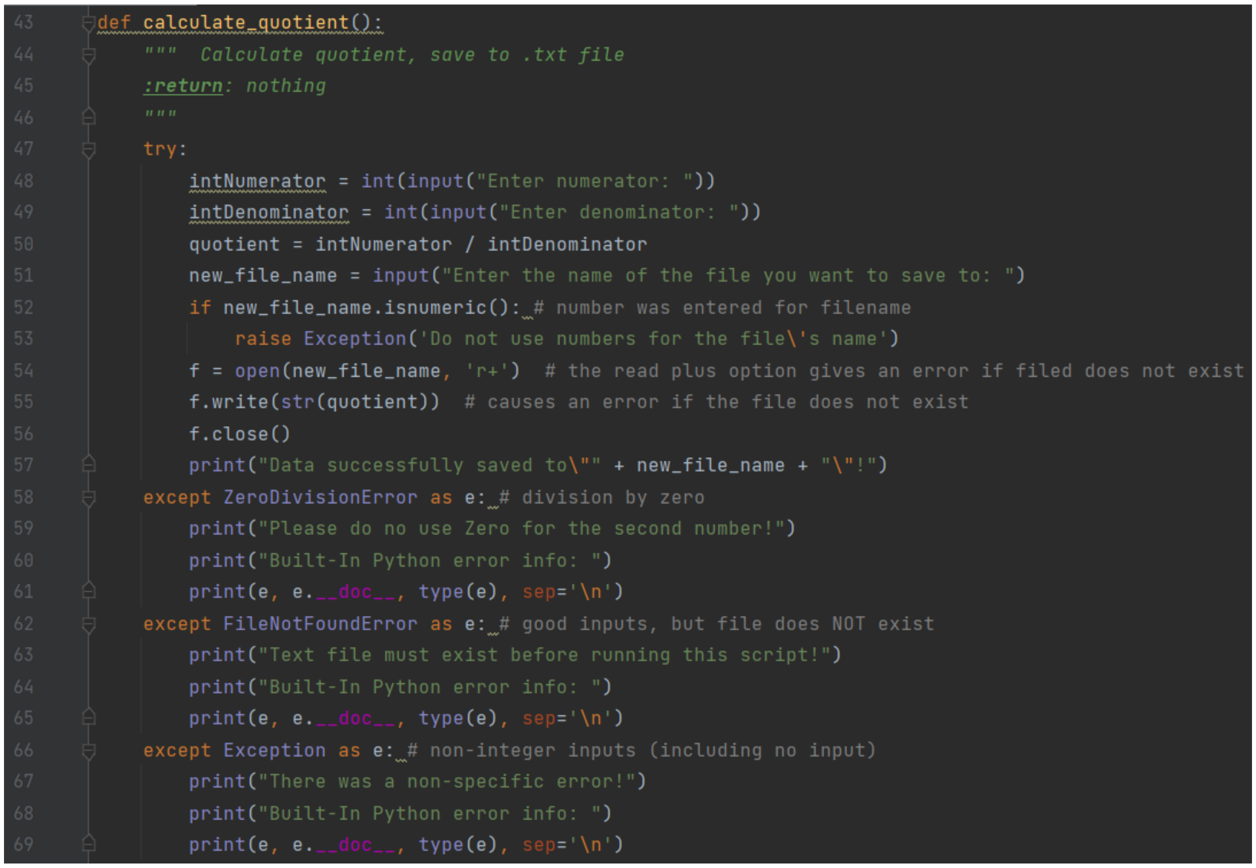
The script contains the following functions:

- save_data_to_file (Figure 3) – Saves data to binary file using "dump" function from pickle module; used in Example #1
- read_data_from_file (Figure 3) – Reads data from binary file using "load" function from pickle module; used in Example #1
- calculate_quotient (Figure 4) – Calculate quotient, save to .txt file; used in Example #2
- output_menu_tasks (Figure 5) – Display a menu of choices to user; used in Example #2
- input_menu_choice (Figure 5) – Gets the menu choice from a user; used in Example #2



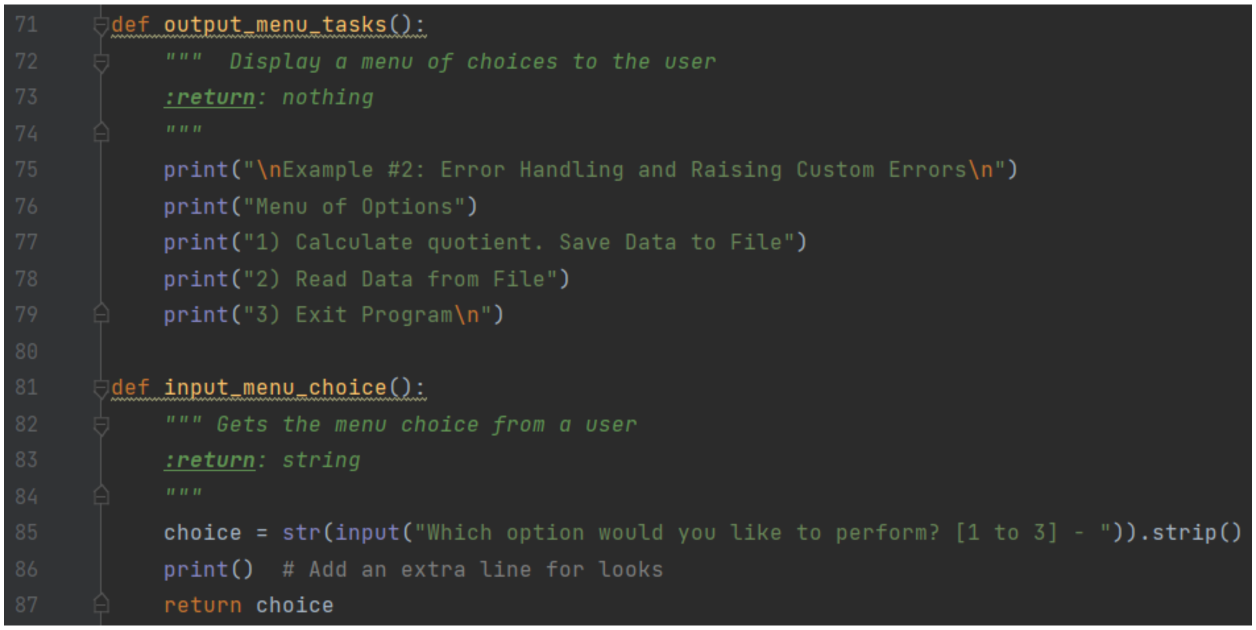
```python
# Processing --------------------------------------- #
def save_data_to_file(file_name, list_of_data):
    """  Save data to binary file
    :return: nothing
    """
    pass
    objFile = open(file_name, "wb+")
    pickle.dump(list_of_data, objFile)
    objFile.close()
    print("Data has been saved to \"" + file_name + "\".")

def read_data_from_file(file_name):
    """  Read data from binary file
    :return: nothing
    """
    pass
    try:
        objFile = open(file_name, "rb+")
        objFileData = pickle.load(objFile)  # load() only loads one row of data.
        objFile.close()
        print("Data stored in \"" + file_name + "\":")
        print(objFileData)
    except FileNotFoundError as e: # file does NOT exist
        print("Data file must exist before running this script!")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
```

*Figure 3. Save and read data from file functions*



```python
def calculate_quotient():
    """  Calculate quotient, save to .txt file
    :return: nothing
    """
    try:
        intNumerator = int(input("Enter numerator: "))
        intDenominator = int(input("Enter denominator: "))
        quotient = intNumerator / intDenominator
        new_file_name = input("Enter the name of the file you want to save to: ")
        if new_file_name.isnumeric(): # number was entered for filename
            raise Exception('Do not use numbers for the file\'s name')
        f = open(new_file_name, 'r+')  # the read plus option gives an error if filed does not exist
        f.write(str(quotient))  # causes an error if the file does not exist
        f.close()
        print("Data successfully saved to\"" + new_file_name + "\"!")
    except ZeroDivisionError as e: # division by zero
        print("Please do no use Zero for the second number!")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
    except FileNotFoundError as e: # good inputs, but file does NOT exist
        print("Text file must exist before running this script!")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
    except Exception as e: # non-integer inputs (including no input)
        print("There was a non-specific error!")
        print("Built-In Python error info: ")
        print(e, e.__doc__, type(e), sep='\n')
```

*Figure 4. calculate quotient function*



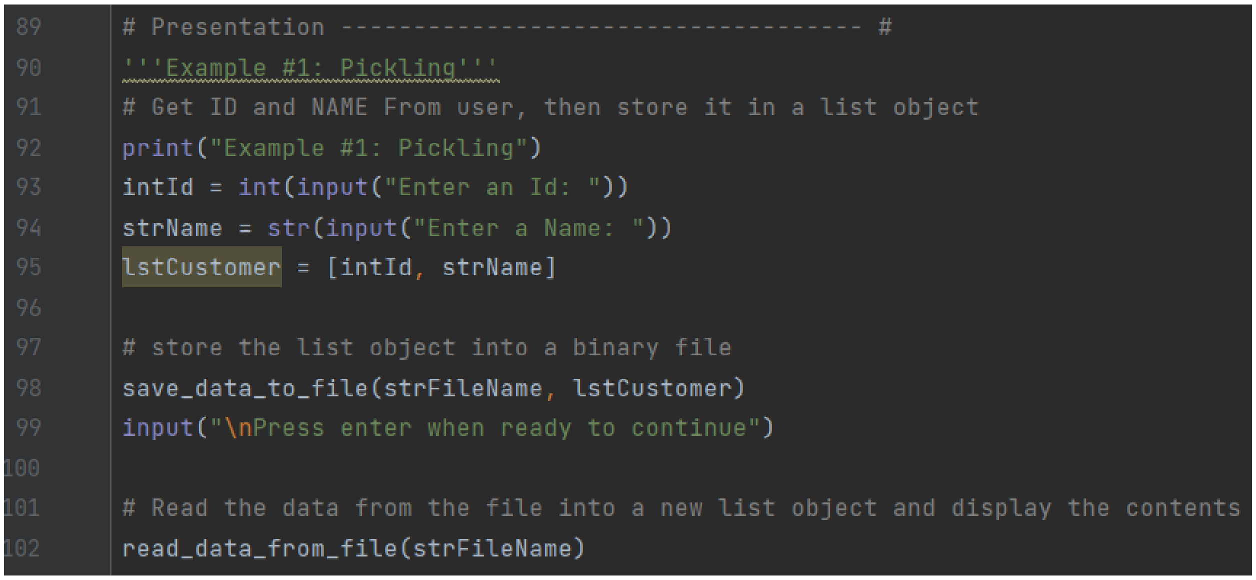
```python
def output_menu_tasks():
    """  Display a menu of choices to the user
    :return: nothing
    """
    print("\nExample #2: Error Handling and Raising Custom Errors\n")
    print("Menu of Options")
    print("1) Calculate quotient. Save Data to File")
    print("2) Read Data from File")
    print("3) Exit Program\n")

def input_menu_choice():
    """ Gets the menu choice from a user
    :return: string
    """
    choice = str(input("Which option would you like to perform? [1 to 3] - ")).strip()
    print()  # Add an extra line for looks
    return choice
```

*Figure 5. display menu and get user menu choice functions*

## Main Body of Script

### Example #1: Pickling

The first part of the script (Figure 6) demonstrates pickling. The user will be prompted to enter an ID number and name. The program will store this information in a list and call the save_data_to_file function to save the data to the file “AppData.dat”. The user will need to press enter to continue (line 99). Then, the read_data_from_file function is called to display the contents of “AppData.dat” in order to verify the data written to the file was correct.

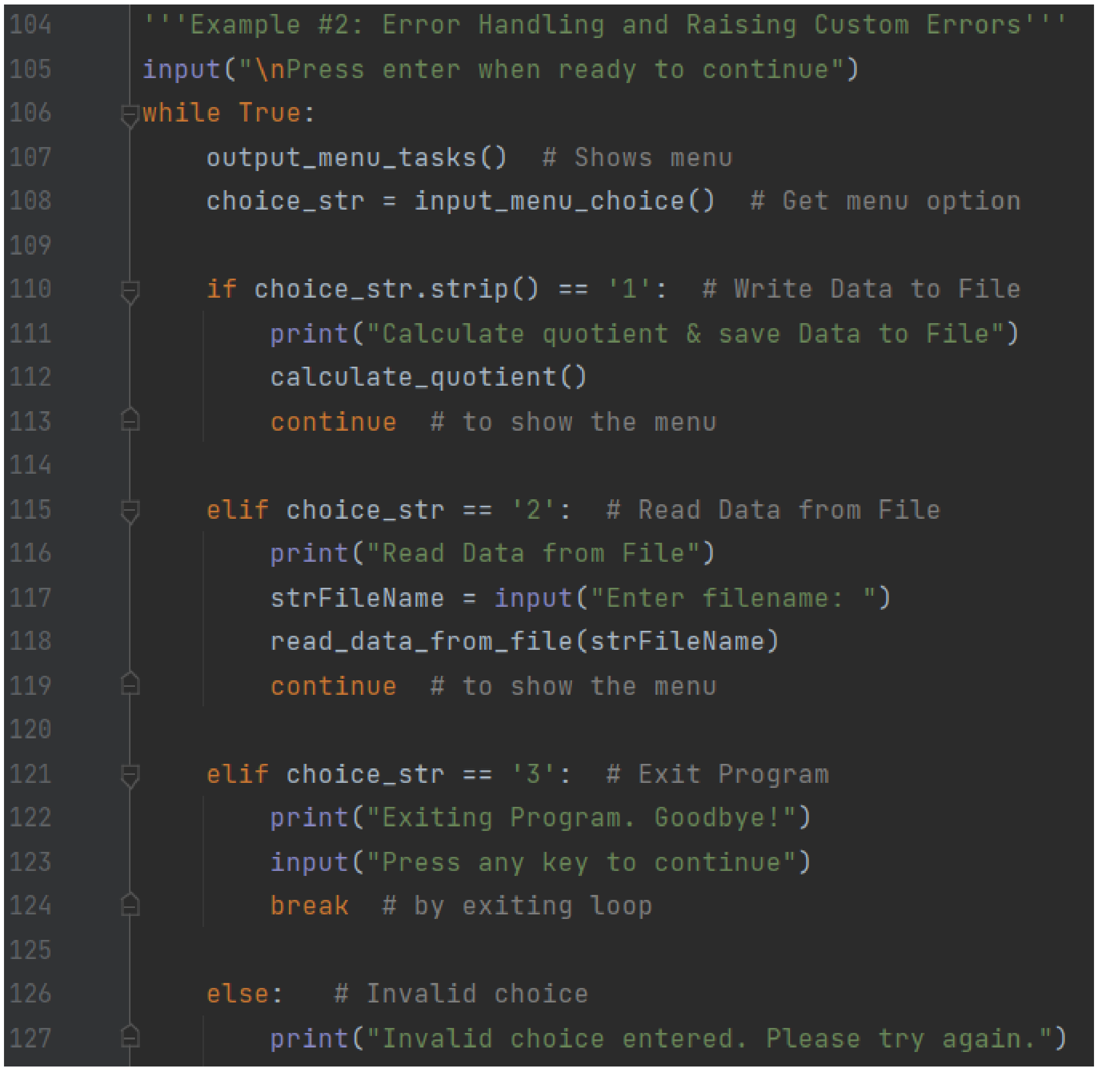


```
 89   # Presentation ------------------------------------ #
 90   '''Example #1: Pickling'''
 91   # Get ID and NAME From user, then store it in a list object
 92   print("Example #1: Pickling")
 93   intId = int(input("Enter an Id: "))
 94   strName = str(input("Enter a Name: "))
 95   lstCustomer = [intId, strName]
 96
 97   # store the list object into a binary file
 98   save_data_to_file(strFileName, lstCustomer)
 99   input("\nPress enter when ready to continue")
100
101   # Read the data from the file into a new list object and display the contents
102   read_data_from_file(strFileName)
```

*Figure 6. Example #1 - Pickling*

### Example #2: Error Handling and Raising Custom Errors

The second part of the script (Figure 7) demonstrates error handling and custom errors. A while loop with a menu is used in order to demonstrate the various error handling situations that could be encountered. The script calls the output_menu_tasks and input_menu_choice functions to display the menu choices to the user and have the user make a selection. The menu options are: 1 – calculate quotient and save data to file, 2 – read data from file, and 3 – exit program. If the user inputs anything else, they will receive an invalid choice message and be prompted to choose again. Option 1 calls the calculate_quotient function and option 2 calls the read_data_from_file function. The “Verify Program Worked” section will go into the various error handling situations that the script implements.

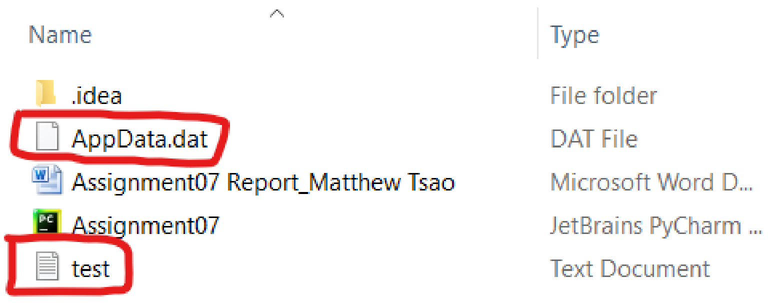


```
'''Example #2: Error Handling and Raising Custom Errors'''
input("\nPress enter when ready to continue")
while True:
    output_menu_tasks()  # Shows menu
    choice_str = input_menu_choice()  # Get menu option

    if choice_str.strip() == '1':  # Write Data to File
        print("Calculate quotient & save Data to File")
        calculate_quotient()
        continue  # to show the menu

    elif choice_str == '2':  # Read Data from File
        print("Read Data from File")
        strFileName = input("Enter filename: ")
        read_data_from_file(strFileName)
        continue  # to show the menu

    elif choice_str == '3':  # Exit Program
        print("Exiting Program. Goodbye!")
        input("Press any key to continue")
        break  # by exiting loop

    else:  # Invalid choice
        print("Invalid choice entered. Please try again.")
```

*Figure 7. Example #2 – Error Handling and Raising Custom Errors*

## Verify Program Worked

The sample files “AppData.dat” and “test.txt” are in my working directory (Figure 8). I used these files to test my program in PyCharm and OS command/shell window.



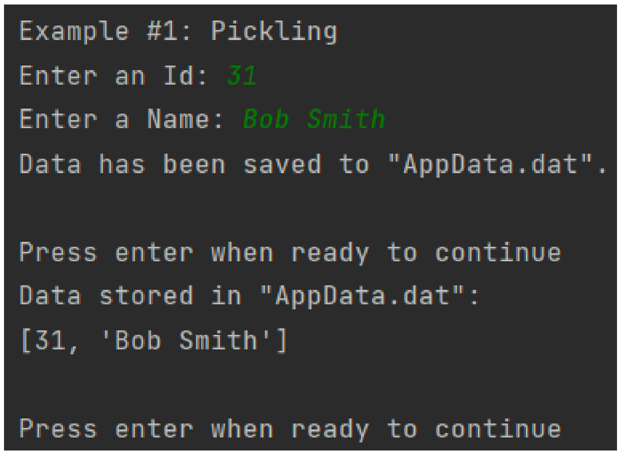
| Name | Type |
|---|---|
| .idea | File folder |
| AppData.dat | DAT File |
| Assignment07 Report_Matthew Tsao | Microsoft Word D... |
| Assignment07 | JetBrains PyCharm ... |
| test | Text Document |

*Figure 8. Working Directory Files*

## Run Program Using PyCharm

**Example #1: Pickling**

User enters an ID and name which are saved to “AppData.dat” (Figure 9). The user presses enter when ready to continue and the script reads the data that was saved to the file. The user presses enter when ready to continue again.

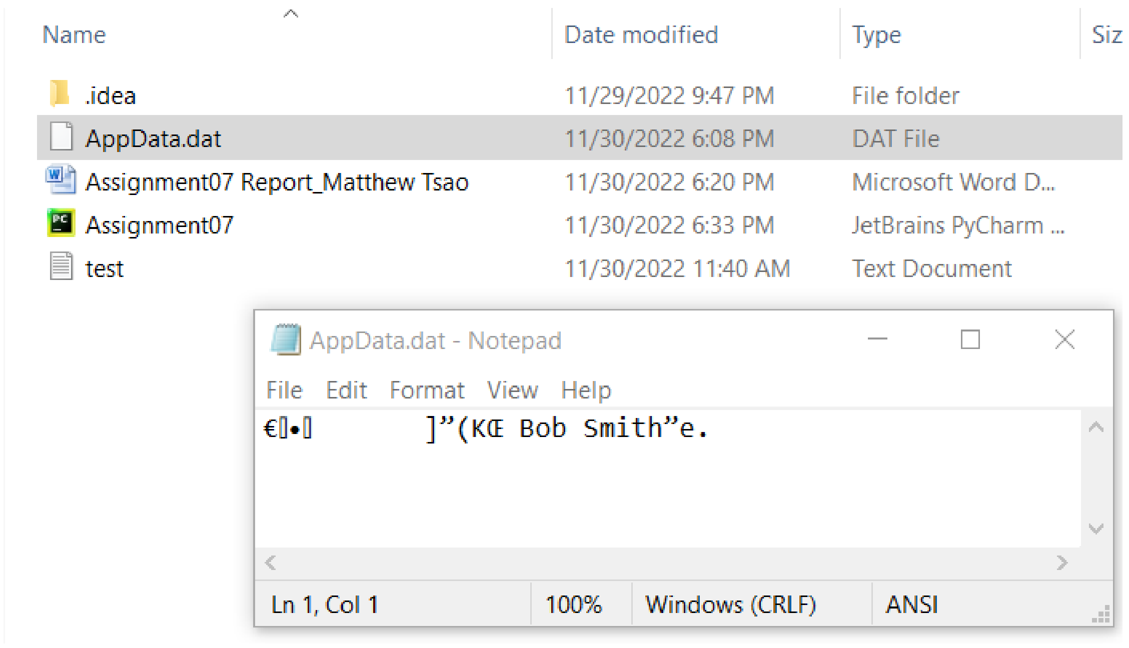


```
Example #1: Pickling
Enter an Id: 31
Enter a Name: Bob Smith
Data has been saved to "AppData.dat".

Press enter when ready to continue
Data stored in "AppData.dat":
[31, 'Bob Smith']

Press enter when ready to continue
```

*Figure 9. Example #1 - Pickling*

## Verify Contents of Output File

Check to verify that the output file was created with data as expected (Figure 10).



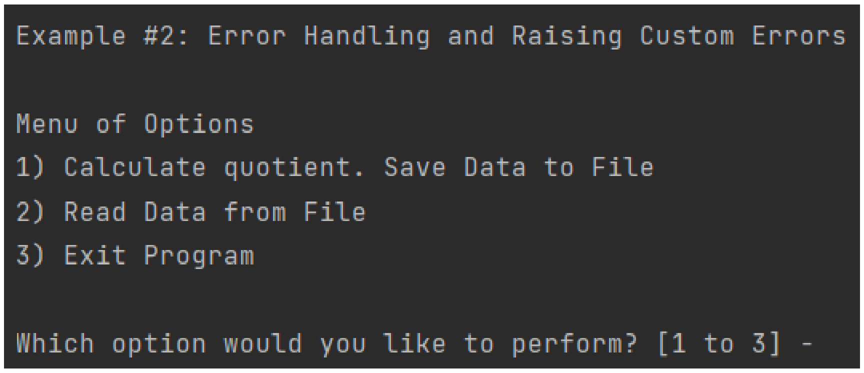
| Name | Date modified | Type | Siz |
|---|---|---|---|
| .idea | 11/29/2022 9:47 PM | File folder | |
| AppData.dat | 11/30/2022 6:08 PM | DAT File | |
| Assignment07 Report_Matthew Tsao | 11/30/2022 6:20 PM | Microsoft Word D... | |
| Assignment07 | 11/30/2022 6:33 PM | JetBrains PyCharm ... | |
| test | 11/30/2022 11:40 AM | Text Document | |

```
€•]”(KŒ Bob Smith”e.
```

*Figure 10. Verification of Output File*

**Example #2: Error Handling and Raising Custom Errors**

The user is presented with a small menu and prompted to make a selection (Figure 11).

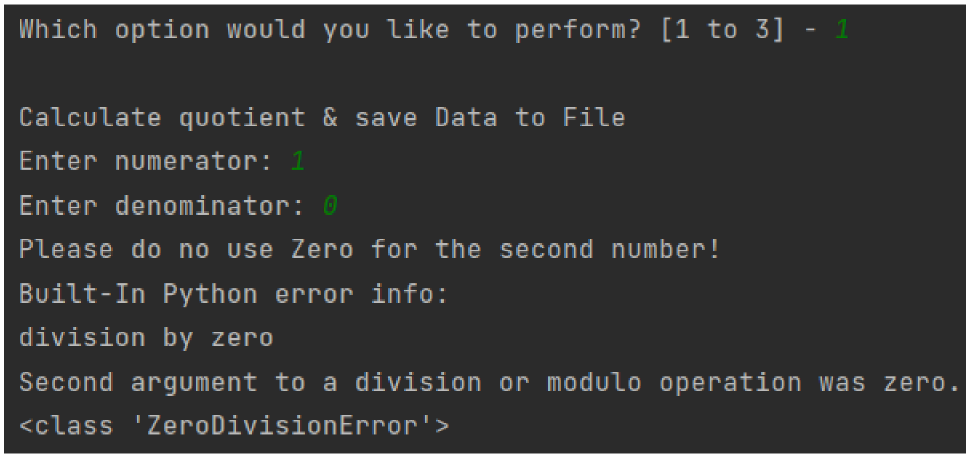


```
Example #2: Error Handling and Raising Custom Errors

Menu of Options
1) Calculate quotient. Save Data to File
2) Read Data from File
3) Exit Program

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

*Figure 11. Example #2 - Error Handling and Raising Custom Errors*

## Error Handling – Division by Zero

The user selected option 1 (Figure 12). The user is prompted to input a value for numerator and denominator. The user entered zero for the denominator, evoking the division by zero handling error.

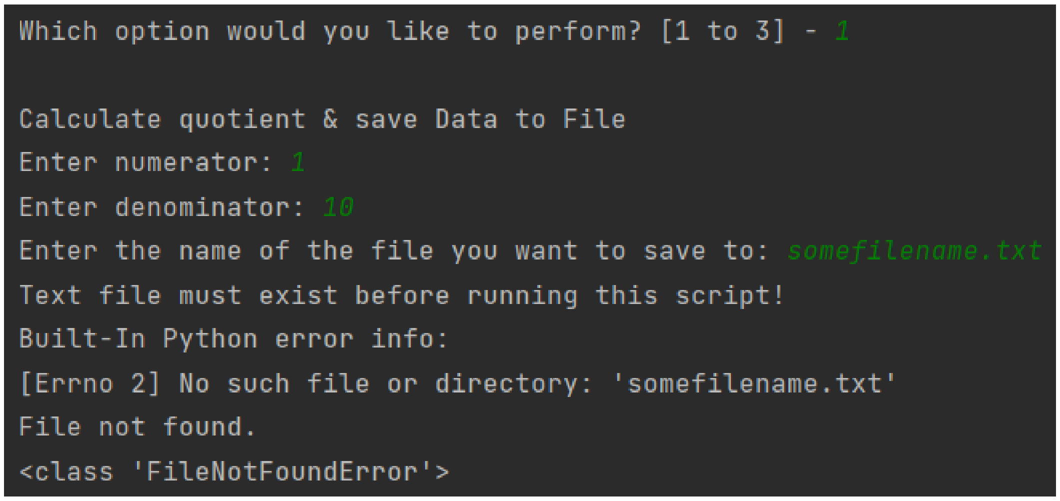


```
Which option would you like to perform? [1 to 3] - 1

Calculate quotient & save Data to File
Enter numerator: 1
Enter denominator: 0
Please do no use Zero for the second number!
Built-In Python error info:
division by zero
Second argument to a division or modulo operation was zero.
<class 'ZeroDivisionError'>
```

*Figure 12. Error Handling - Division by Zero*

## Error Handling – File Not Found

The user selected option 1 (Figure 13). The user is prompted to input a value for numerator and denominator. Then, the user is asked to enter a filename to save the entered data to. For simplicity, the script will evoke a file not found error if the entered filename is not in the working directory.

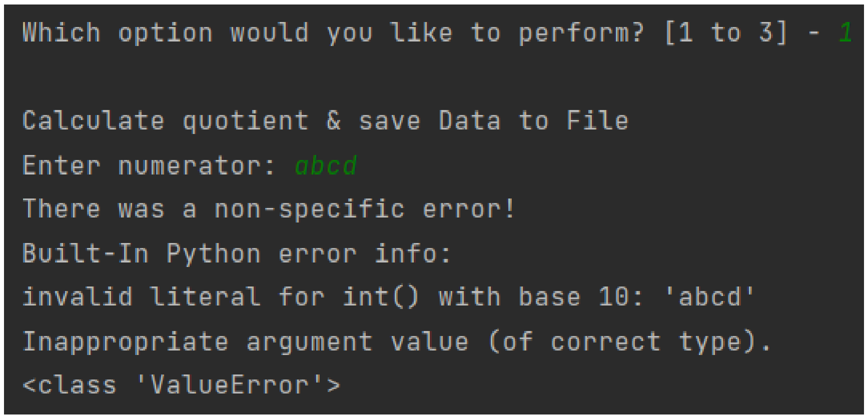


```
Which option would you like to perform? [1 to 3] - 1

Calculate quotient & save Data to File
Enter numerator: 1
Enter denominator: 10
Enter the name of the file you want to save to: somefilename.txt
Text file must exist before running this script!
Built-In Python error info:
[Errno 2] No such file or directory: 'somefilename.txt'
File not found.
<class 'FileNotFoundError'>
```

*Figure 13. Error Handling – File Not Found*

## Error Handling – Non-Specific Errors

The user selected option 1 (Figure 14). The user is prompted to input a value for the numerator. The user did NOT enter an integer value, evoking the non-specific error handling error.

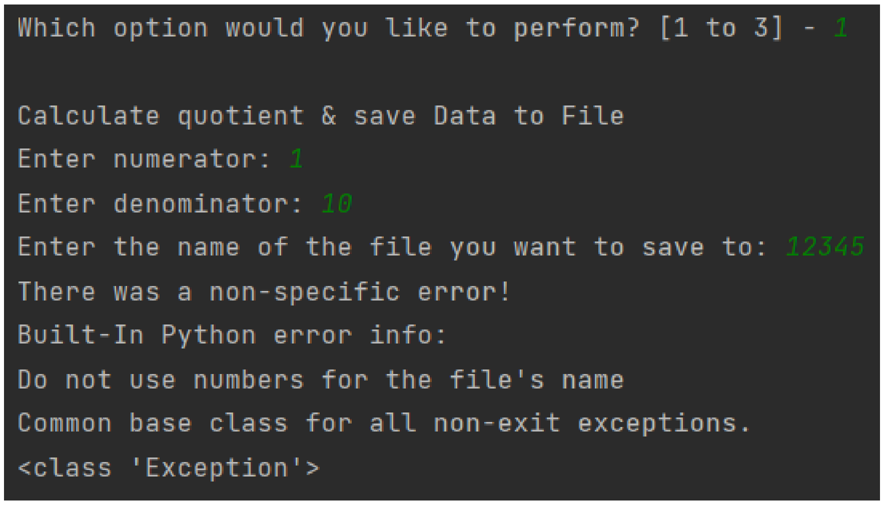


```
Which option would you like to perform? [1 to 3] - 1

Calculate quotient & save Data to File
Enter numerator: abcd
There was a non-specific error!
Built-In Python error info:
invalid literal for int() with base 10: 'abcd'
Inappropriate argument value (of correct type).
<class 'ValueError'>
```

*Figure 14. Error Handling – Non-Specific Error*

## Raised Custom Error – Numeric Filename Search

The user selected option 1 (Figure 15). The user is prompted to input a value for numerator and denominator. Then, the user is asked to enter a filename to save the entered data to. For simplicity, the script will evoke a file not found error if the entered filename is not in the working directory. Since the user tried to enter a number for the filename, the raised custom error that numbers can NOT be used for the filename was evoked. The script also evoked the non-specific error handling error.

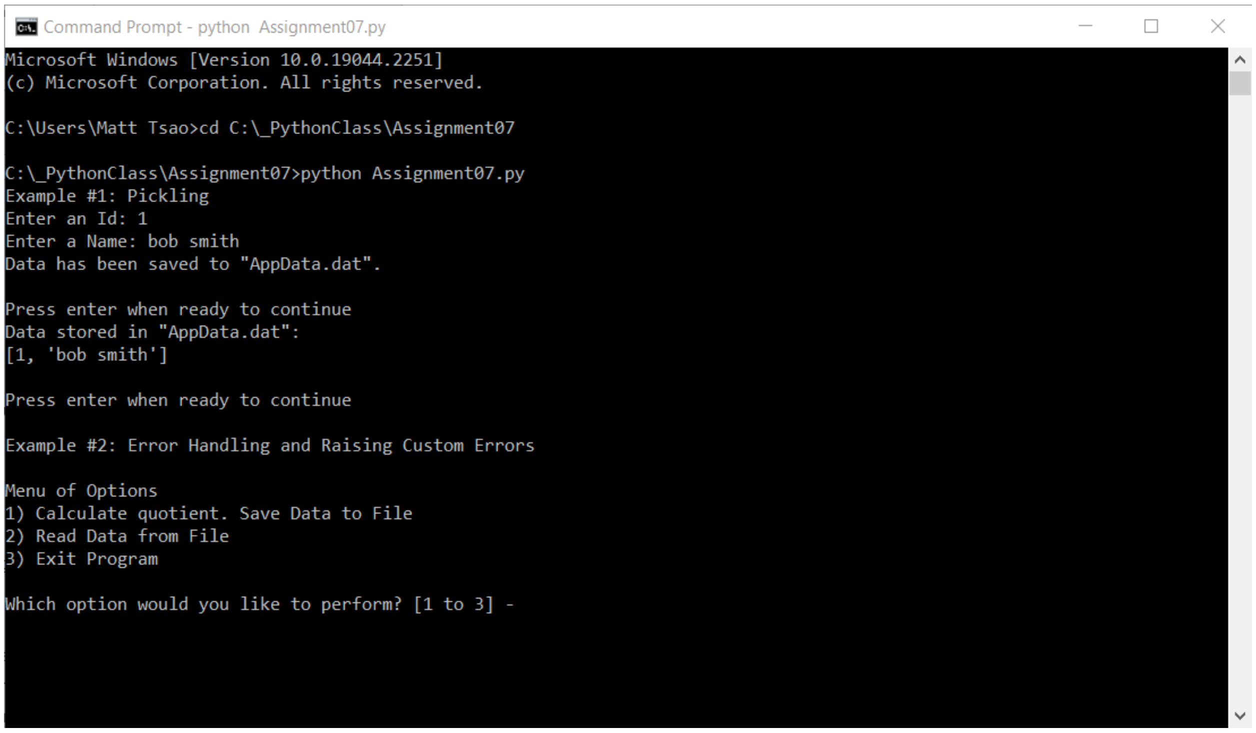


```
Which option would you like to perform? [1 to 3] - 1

Calculate quotient & save Data to File
Enter numerator: 1
Enter denominator: 10
Enter the name of the file you want to save to: 12345
There was a non-specific error!
Built-In Python error info:
Do not use numbers for the file's name
Common base class for all non-exit exceptions.
<class 'Exception'>
```

*Figure 15. Raised Custom Error - Numeric Filename Search*

### Run Program Using OS command/shell window

Snip of Assignment07.py running using shell window (Figure 16).



```
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Matt Tsao>cd C:\_PythonClass\Assignment07

C:\_PythonClass\Assignment07>python Assignment07.py
Example #1: Pickling
Enter an Id: 1
Enter a Name: bob smith
Data has been saved to "AppData.dat".

Press enter when ready to continue
Data stored in "AppData.dat":
[1, 'bob smith']

Press enter when ready to continue

Example #2: Error Handling and Raising Custom Errors

Menu of Options
1) Calculate quotient. Save Data to File
2) Read Data from File
3) Exit Program

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

*Figure 16. Run Assignment07.py in shell window*

## Summary

The Assignment07.py program demonstrated the following concepts:

- Exception handling
    - Division by zero
    - File doesn’t exist

    - Non-specific errors
- Pickling/working with binary files
- Raising custom errors
- Creating advanced GitHub pages