Deja Monet

Randal Root

Foundations of Python

24 August 2022

GitHub

Assignment 07: Files & Exceptions

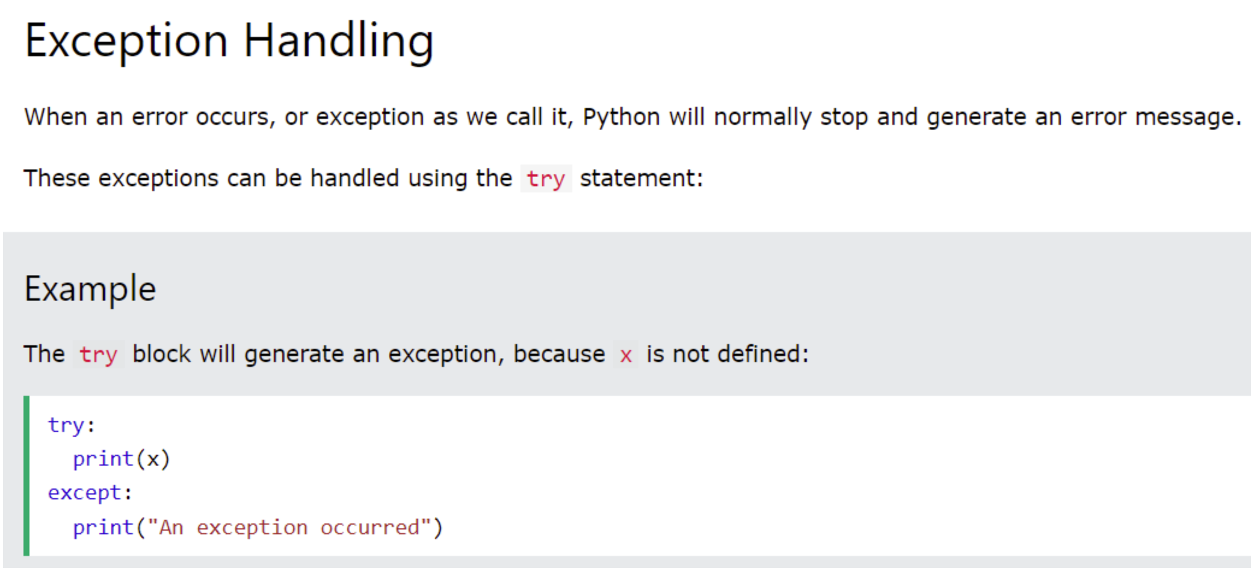
Introduction

In this assignment, I will review Python exceptions or error handling and pickling. I will also review defining and calling custom errors as functions.

Completing the Assignment: Researching Exception Handling

This assignment directed students to learn about exception handling and Pickling in Python. My first step was to research what exception handling was, and choose an appropriate script that demonstrated this issue.

The first resource I used to research exception handling was w3schools.com. This resource showed that exceptions can be handled using the **try** statement (**Figure 1**).


Exception Handling

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

These exceptions can be handled using the `try` statement:

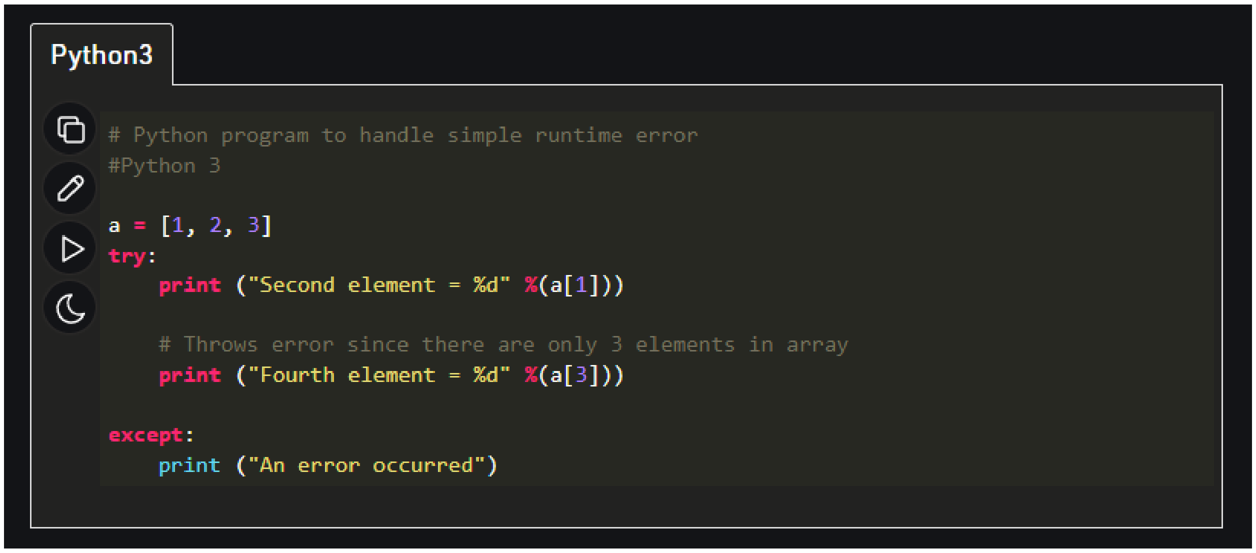
Example

The `try` block will generate an exception, because `x` is not defined:

```
try:
  print(x)
except:
  print("An exception occurred")
```

Figure 1. A description of Exception Handling from w3schools.com

I then checked geeksforgeeks.org to see more examples of exception handling (**Figure 2**).



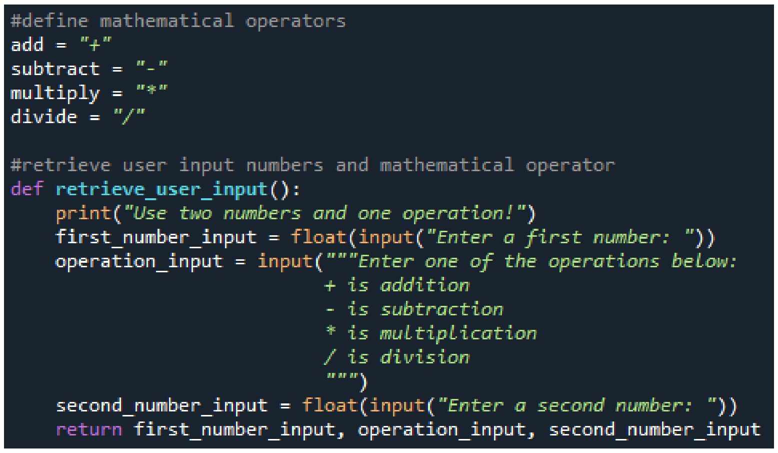
```
# Python program to handle simple runtime error
#Python 3

a = [1, 2, 3]
try:
    print ("Second element = %d" %(a[1]))

    # Throws error since there are only 3 elements in array
    print ("Fourth element = %d" %(a[3]))

except:
    print ("An error occurred")
```

Figure 2. An example of Exception Handling from geeksforgeeks.org

From these two resources, I learned that a script can be written to catch built-in Python exceptions, or a programmer can create their own custom exceptions. I decided that to demonstrate error handling, I would write a script that performed as a basic calculator. In order to elevate this script from the previous assignment in this course in which we used basic mathematical operations, I decided I would use classes and functions in order to retrieve the user's input, process data, and define custom errors.

Completing the Assignment: Writing a Script

I began by defining the mathematical operators add, subtract, multiply, and divide as strings, then defined a function to retrieve the user's first number, choice of mathematical operator, and second number as the function **retrieve_user_input**. This function returns the user's input so that they can be used in other functions (**Figure 3**).

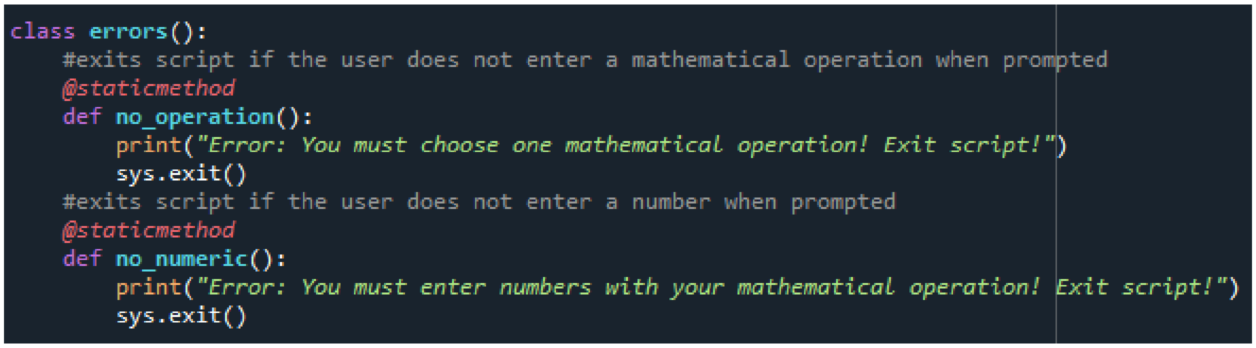


```
#define mathematical operators
add = "+"
subtract = "-"
multiply = "*"
divide = "/"

#retrieve user input numbers and mathematical operator
def retrieve_user_input():
    print("Use two numbers and one operation!")
    first_number_input = float(input("Enter a first number: "))
    operation_input = input("""Enter one of the operations below:
                            + is addition
                            - is subtraction
                            * is multiplication
                            / is division
                            """)
    second_number_input = float(input("Enter a second number: "))
    return first_number_input, operation_input, second_number_input
```

Figure 3. The mathematical operators are defined, and user input is retrieved in a function.

I decided that potential errors from the user input could be if the user did not enter a mathematical operation when prompted, or did not enter a number when prompted. I defined the class

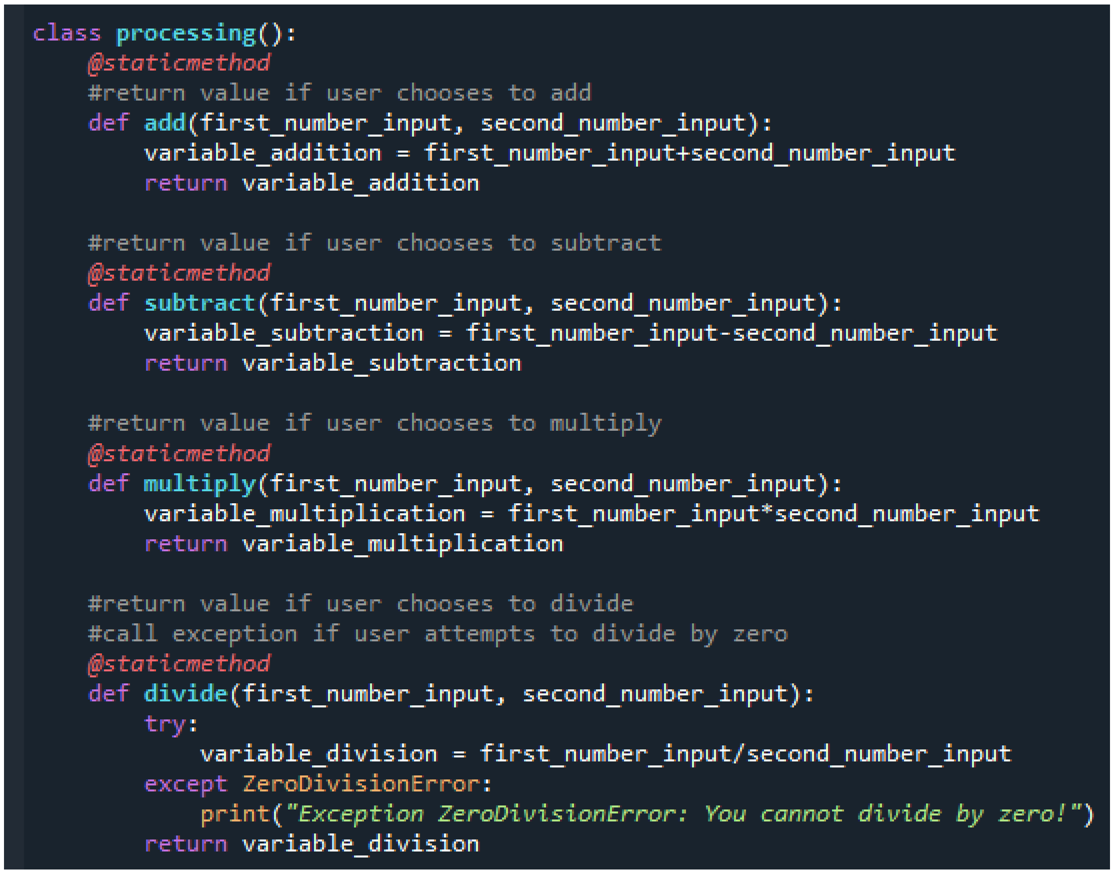
errors and created the functions **no_operation()** and **no_numeric()** to capture these respective issues (**Figure 4**).



```
class errors():
    #exits script if the user does not enter a mathematical operation when prompted
    @staticmethod
    def no_operation():
        print("Error: You must choose one mathematical operation! Exit script!")
        sys.exit()
    #exits script if the user does not enter a number when prompted
    @staticmethod
    def no_numeric():
        print("Error: You must enter numbers with your mathematical operation! Exit script!")
        sys.exit()
```

Figure 4. Custom errors are raised if the user does not enter the correct input when prompted.

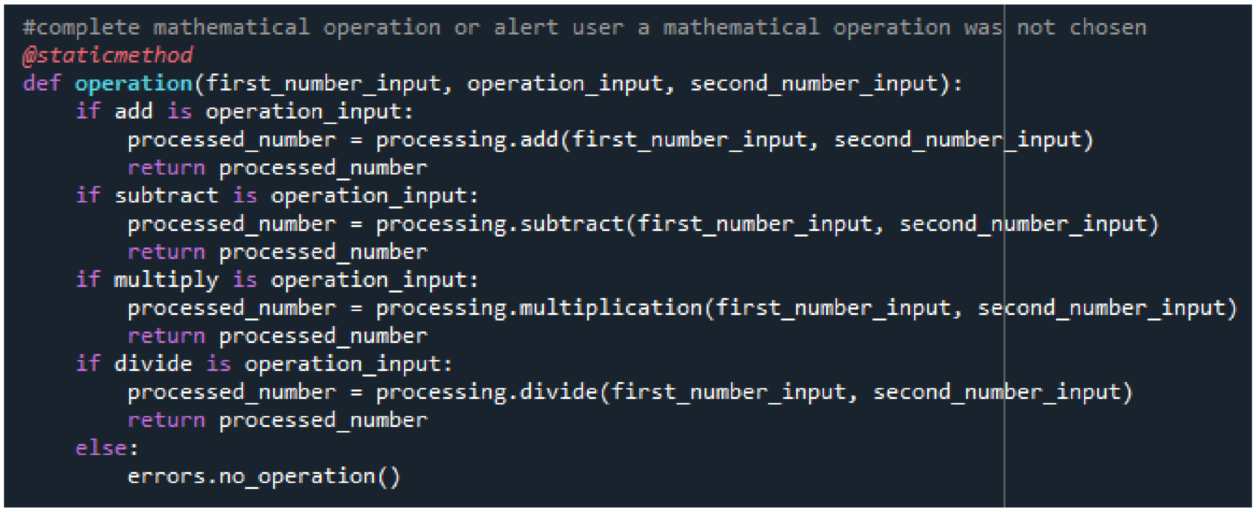
I then defined a **processing** class that completes the mathematical operation as chosen by the user, with each mathematical operation defined as its own function. The **divide** function has the potential to raise the **ZeroDivisionError** exception, in which the user is alerted that they cannot divide by zero (**Figure 5**).



```
class processing():
    @staticmethod
    #return value if user chooses to add
    def add(first_number_input, second_number_input):
        variable_addition = first_number_input+second_number_input
        return variable_addition

    #return value if user chooses to subtract
    @staticmethod
    def subtract(first_number_input, second_number_input):
        variable_subtraction = first_number_input-second_number_input
        return variable_subtraction

    #return value if user chooses to multiply
    @staticmethod
    def multiply(first_number_input, second_number_input):
        variable_multiplication = first_number_input*second_number_input
        return variable_multiplication

    #return value if user chooses to divide
    #call exception if user attempts to divide by zero
    @staticmethod
    def divide(first_number_input, second_number_input):
        try:
            variable_division = first_number_input/second_number_input
        except ZeroDivisionError:
            print("Exception ZeroDivisionError: You cannot divide by zero!")
        return variable_division
```

Figure 5. The ZeroDivisionError exception is raised if the user attempts to divide by zero.

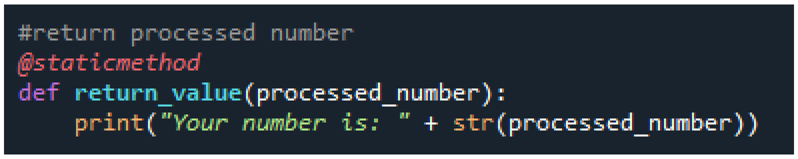
The processing class also contains an **operation** function, which calls the mathematical operation function that the user chose. This function also returns an error message if the user did not choose a mathematical operation when prompted (**Figure 6**).



```
#complete mathematical operation or alert user a mathematical operation was not chosen
@staticmethod
def operation(first_number_input, operation_input, second_number_input):
    if add is operation_input:
        processed_number = processing.add(first_number_input, second_number_input)
        return processed_number
    if subtract is operation_input:
        processed_number = processing.subtract(first_number_input, second_number_input)
        return processed_number
    if multiply is operation_input:
        processed_number = processing.multiplication(first_number_input, second_number_input)
        return processed_number
    if divide is operation_input:
        processed_number = processing.divide(first_number_input, second_number_input)
        return processed_number
    else:
        errors.no_operation()
```

Figure 6. The operation function calls the mathematical operation that the user chose, or returns an error message.

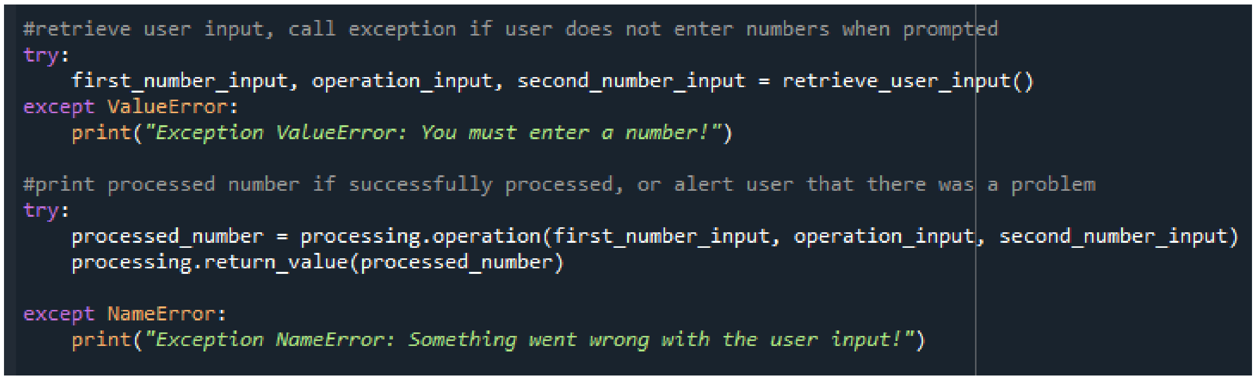
The processing class contains one final function that returns the processed number if the user entered two numbers and a mathematical operation correctly when prompted (**Figure 7**).



```
#return processed number
@staticmethod
def return_value(processed_number):
    print("Your number is: " + str(processed_number))
```

Figure 7. The processed number is returned to the user.

I then added two **try/except** statements in order to retrieve the user input and to process the user input. The first of these returns a message if a **ValueError** exception is raised, if the user does not enter a number when prompted. The second of these returns a message if a **NameError** exception is raised, and gives a general statement that there was a problem with the user input (**Figure 8**).



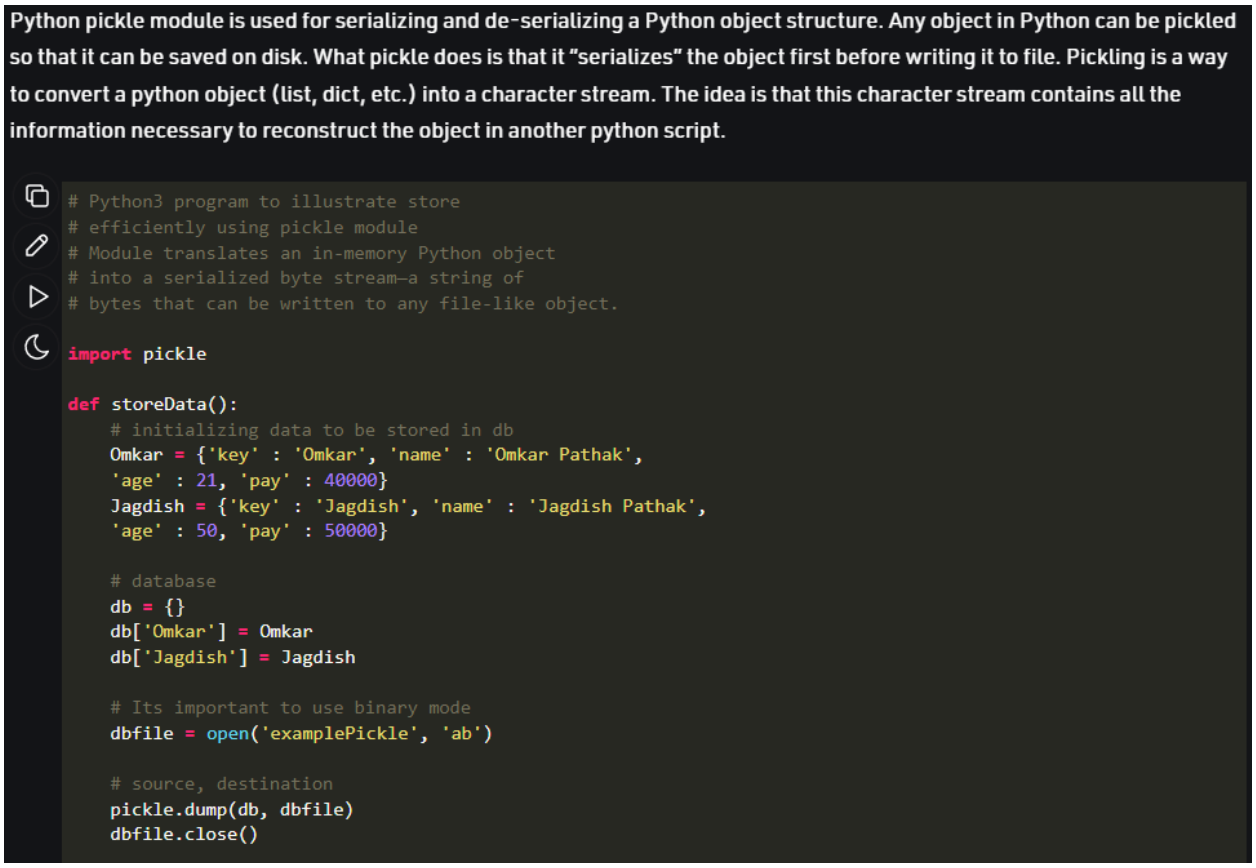
```
#retrieve user input, call exception if user does not enter numbers when prompted
try:
    first_number_input, operation_input, second_number_input = retrieve_user_input()
except ValueError:
    print("Exception ValueError: You must enter a number!")

#print processed number if successfully processed, or alert user that there was a problem
try:
    processed_number = processing.operation(first_number_input, operation_input, second_number_input)
    processing.return_value(processed_number)

except NameError:
    print("Exception NameError: Something went wrong with the user input!")
```

Figure 8. Try/Except statements that return error messages if specific conditions are met.

With the research for exception handling completed and a script written that demonstrated this, I moved on to researching pickling.

Completing the Assignment: Researching Pickling

The first resource I checked was geeksforgeeks.org. I learned that pickle is a module in Python that is used to serialize an object before it is written to a file (**Figure 9**).



Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it "serializes" the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

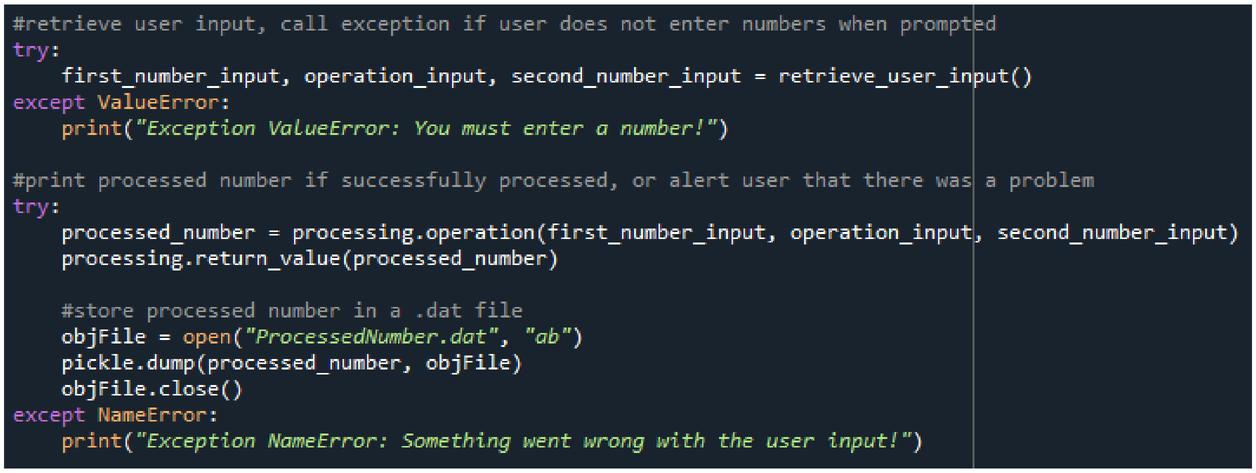
```
# Python3 program to illustrate store
# efficiently using pickle module
# Module translates an in-memory Python object
# into a serialized byte stream—a string of
# bytes that can be written to any file-like object.

import pickle

def storeData():
    # initializing data to be stored in db
    Omkar = {'key' : 'Omkar', 'name' : 'Omkar Pathak',
    'age' : 21, 'pay' : 40000}
    Jagdish = {'key' : 'Jagdish', 'name' : 'Jagdish Pathak',
    'age' : 50, 'pay' : 50000}

    # database
    db = {}
    db['Omkar'] = Omkar
    db['Jagdish'] = Jagdish

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()
```

Figure 9. A description of Pickling from geeksforgeeks.org.

I thought that I understood Pickling from reading the geeksforgeeks.org article, so I did not research any additional websites on this topic. In order to demonstrate Pickling in my script, I would need to import the pickle module and then write an object to a file using the pickle module.

Completing the Assignment: Editing the Script

In order to demonstrate pickling in my script, I first imported the pickle module using **import pickle**. I decided that I would edit the last **try/except** statement such that if the user input was successfully processed, this output would be written to a .dat file (**Figure 10**).



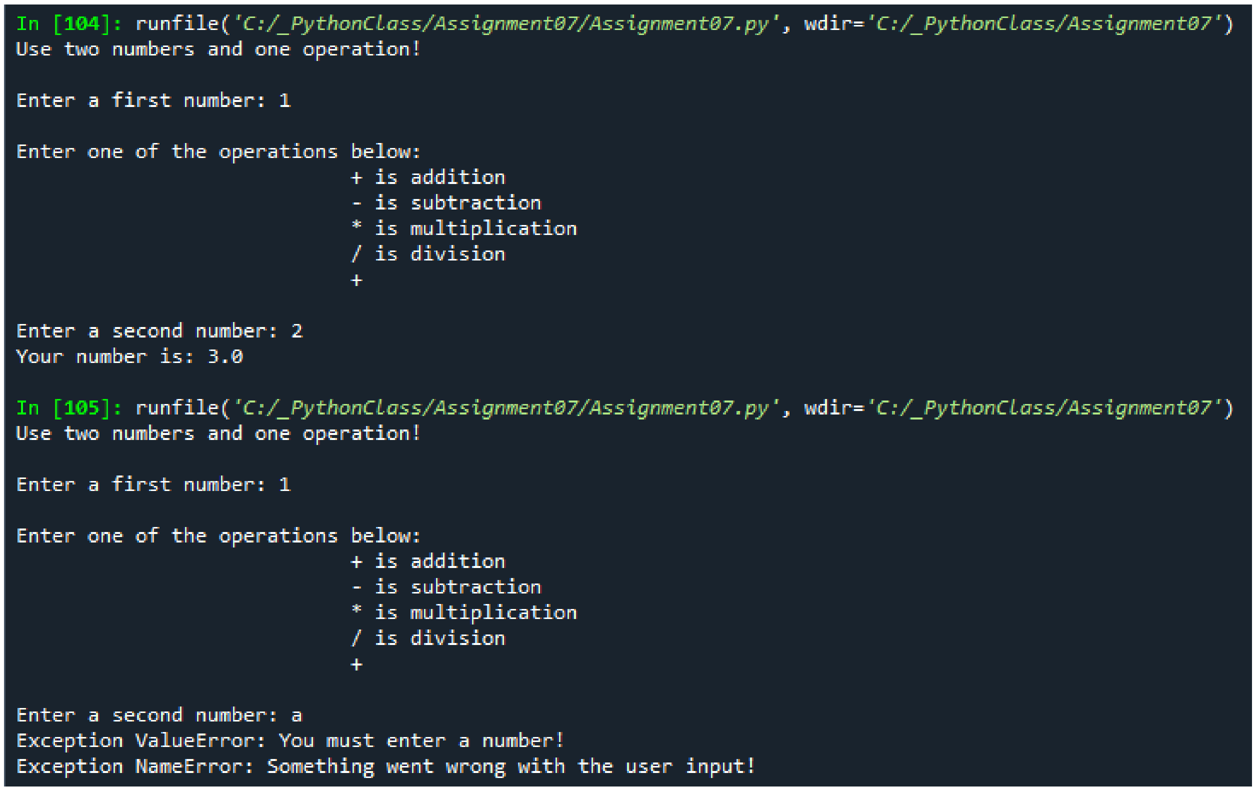
```python
#retrieve user input, call exception if user does not enter numbers when prompted
try:
    first_number_input, operation_input, second_number_input = retrieve_user_input()
except ValueError:
    print("Exception ValueError: You must enter a number!")

#print processed number if successfully processed, or alert user that there was a problem
try:
    processed_number = processing.operation(first_number_input, operation_input, second_number_input)
    processing.return_value(processed_number)

    #store processed number in a .dat file
    objFile = open("ProcessedNumber.dat", "ab")
    pickle.dump(processed_number, objFile)
    objFile.close()
except NameError:
    print("Exception NameError: Something went wrong with the user input!")
```

Figure 10. The processed number is written to a .dat file using the pickle module.

With the script demonstrating both error handling and pickling, the script was completed, and it was time to begin testing the script in an IDE and Anaconda Prompt.

Completing the Assignment: Testing the Script

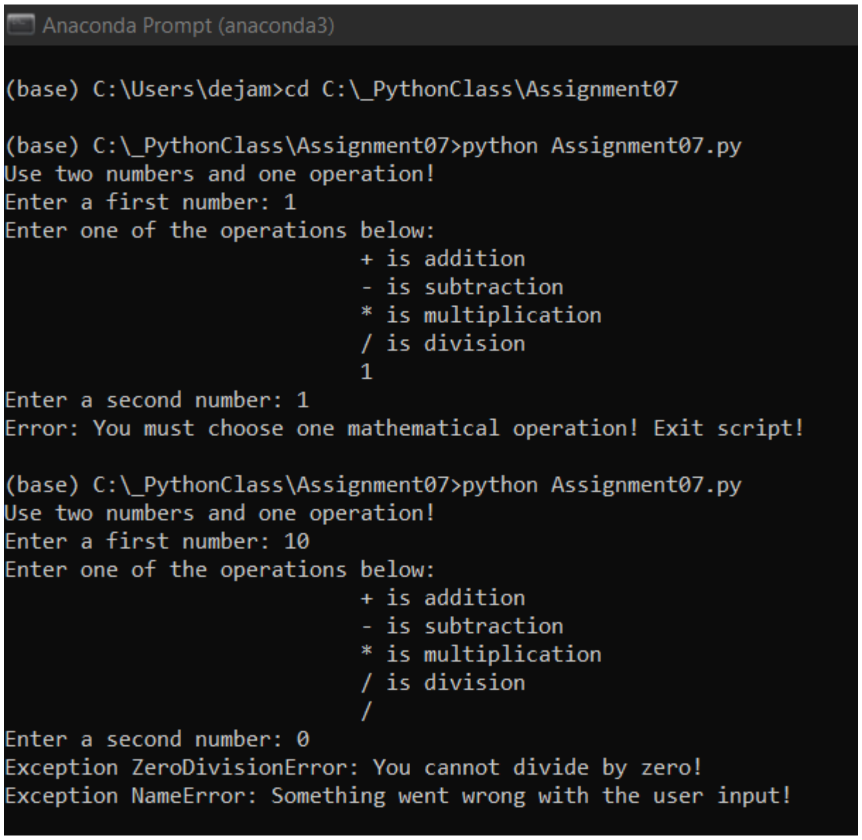
I wrote this script in Spyder, and thus decided to use Spyder as my IDE for testing the script. The script prompts the user for a first number, a mathematical operation, and then a second number. I began by entering an acceptable input: 1+2. The script correctly returned the answer as 3.0. I then entered “a” as my second number, and the script correctly returned a ValueError and NameError exception (**Figure 11**).



```
In [104]: runfile('C:/_PythonClass/Assignment07/Assignment07.py', wdir='C:/_PythonClass/Assignment07')
Use two numbers and one operation!

Enter a first number: 1

Enter one of the operations below:
                            + is addition
                            - is subtraction
                            * is multiplication
                            / is division
                            +

Enter a second number: 2
Your number is: 3.0

In [105]: runfile('C:/_PythonClass/Assignment07/Assignment07.py', wdir='C:/_PythonClass/Assignment07')
Use two numbers and one operation!

Enter a first number: 1

Enter one of the operations below:
                            + is addition
                            - is subtraction
                            * is multiplication
                            / is division
                            +

Enter a second number: a
Exception ValueError: You must enter a number!
Exception NameError: Something went wrong with the user input!
```

Figure 11. Testing the script in Spyder.

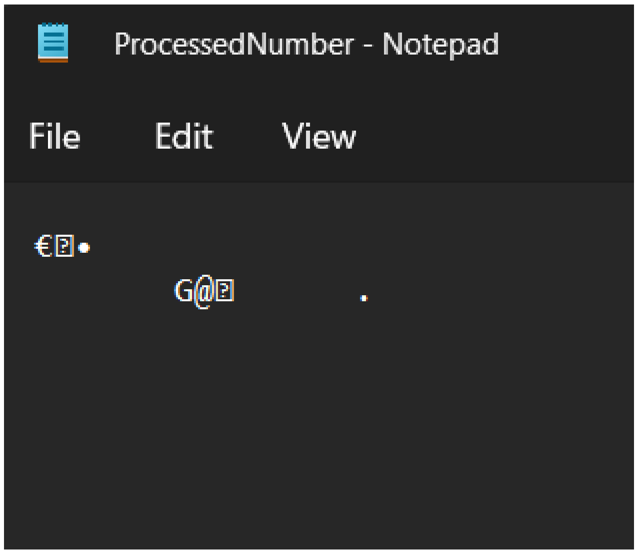
I then needed to test the script in Anaconda Prompt. In Anaconda Prompt, I set the working directory to the correct location, then called the script using **python**. I decided to enter 1 as the first number, operation, and second number, and the script correctly returned one of my custom errors, as there was no operation entered. I then entered 10/0, and the script correctly returned the **ZeroDivisionError** and **NameError** (**Figure 12**).



```
(base) C:\Users\dejam>cd C:\_PythonClass\Assignment07

(base) C:\_PythonClass\Assignment07>python Assignment07.py
Use two numbers and one operation!
Enter a first number: 1
Enter one of the operations below:
                            + is addition
                            - is subtraction
                            * is multiplication
                            / is division
                            1
Enter a second number: 1
Error: You must choose one mathematical operation! Exit script!

(base) C:\_PythonClass\Assignment07>python Assignment07.py
Use two numbers and one operation!
Enter a first number: 10
Enter one of the operations below:
                            + is addition
                            - is subtraction
                            * is multiplication
                            / is division
                            /
Enter a second number: 0
Exception ZeroDivisionError: You cannot divide by zero!
Exception NameError: Something went wrong with the user input!
```

Figure 12. Testing the script in Anaconda Prompt.

Then, in Anaconda Prompt, I entered 3+2, and the script correctly returned 5.0. I then checked to see if a .dat file had been written to the working directory, and found that this had also been successful, so I opened the file in Notepad and found that it had been properly serialized (**Figure 13**).



```
€⍰•
             G@⍰          .
```

Figure 13. The ProcessedNumber.dat file open in Notepad.

With the script able to demonstrate error handling, pickling, and tested in an IDE and Anaconda Prompt, the assignment was complete.

Summary

In this assignment, I reviewed errors and error handling and writing serialized data using the pickle module. I also reviewed printing custom messages when built-in Python Exceptions are triggered in a script.