Grant Kumataka

August 24th, 2021

Foundations of Programming: Python

Assignment 07

https://github.com/Kumatakasan/IntroToProg-Python-Mod07

Assignment 07

Introduction

In This document I will be going over the process I used in completing assignment 07 of the course "Foundations of Programming: Python" by instructor Randall Root. The assignment is to create a custom program showing use of "pickle" and binary formats. Also including creating own exception handling.

Pickling

Pickling is a way to serialize and deserialize data. It works with binary files. The major advantage to using pickling is that it doesn't require any other code to do this serialization. Binary files are harder to read, although pickling does not encrypt data. Binary files are typically smaller in file size, so maybe an advantage when working with large datasets.

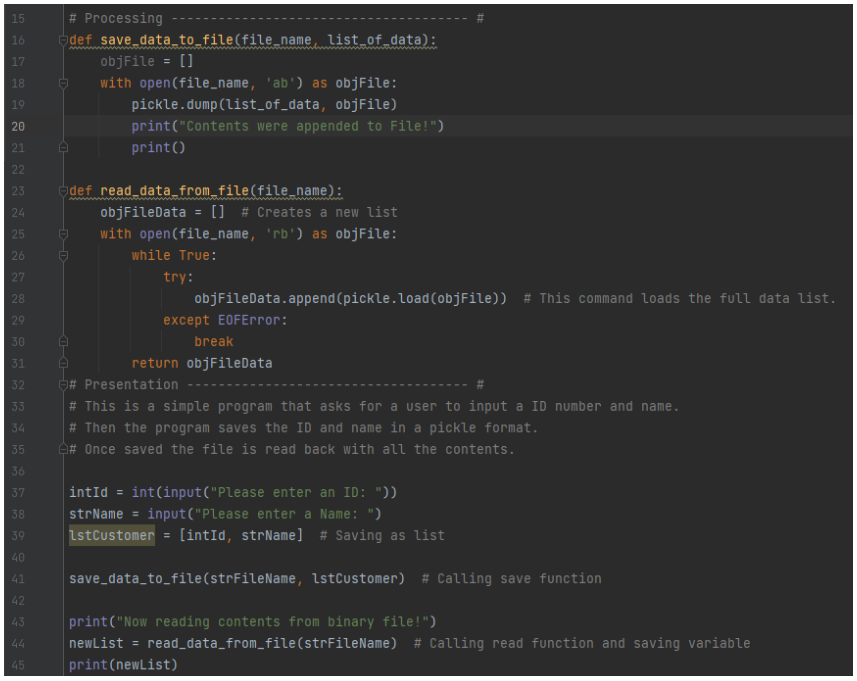
Try/Exceptions

This type of code is used to keep the program running and giving user feedback if something is wrong with the code. This could be because the input is not an integer. This is so that the code can run without failing. Custom exceptions can be made to give the user more feedback about the problem.

Creating the Program

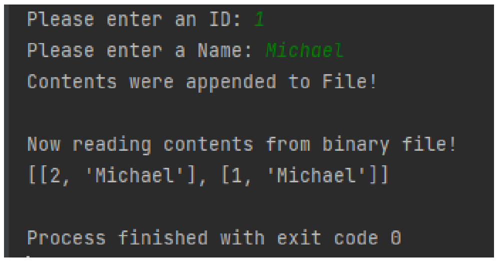
We were tasked with creating our own program with no specific requirements. I decided to follow the Lab7.1_starter.py for the main pickling demo. This sample asked to take in an ID number and assign it to a name. Then I moved onto another script for an easier demo of exception handling in python using the previous assignment of doing some math. This involved making sure the inputs are integers and catching the dividing by zero error.

Figure 1 shows some code for the pickling demo. Figure 2 shows the output of the pickling demo code. Figure 3 shows the math processing code to check the input and to do the math. Figure 4 shows the main part of the math script. Figure 5 shows the math script having a successful run and output some math. Figure 6 shows the response when the input is not a number. Figure 7 shows the response when division by zero happens.



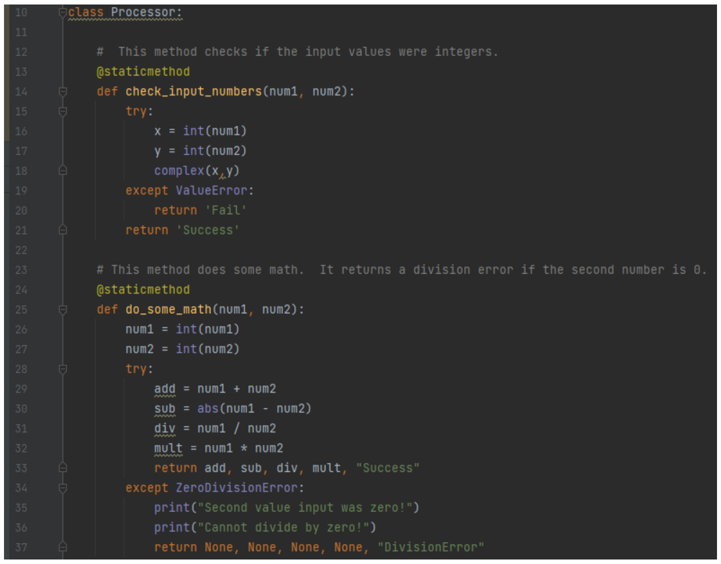
```python
# Processing ------------------------------------- #
def save_data_to_file(file_name, list_of_data):
    objFile = []
    with open(file_name, 'ab') as objFile:
        pickle.dump(list_of_data, objFile)
        print("Contents were appended to File!")
        print()

def read_data_from_file(file_name):
    objFileData = []  # Creates a new list
    with open(file_name, 'rb') as objFile:
        while True:
            try:
                objFileData.append(pickle.load(objFile))  # This command loads the full data list.
            except EOFError:
                break
        return objFileData
# Presentation ----------------------------------- #
# This is a simple program that asks for a user to input a ID number and name.
# Then the program saves the ID and name in a pickle format.
# Once saved the file is read back with all the contents.

intId = int(input("Please enter an ID: "))
strName = input("Please enter a Name: ")
lstCustomer = [intId, strName]  # Saving as list

save_data_to_file(strFileName, lstCustomer)  # Calling save function

print("Now reading contents from binary file!")
newList = read_data_from_file(strFileName)  # Calling read function and saving variable
print(newList)
```

Figure 1. Snippet of Pickling code.



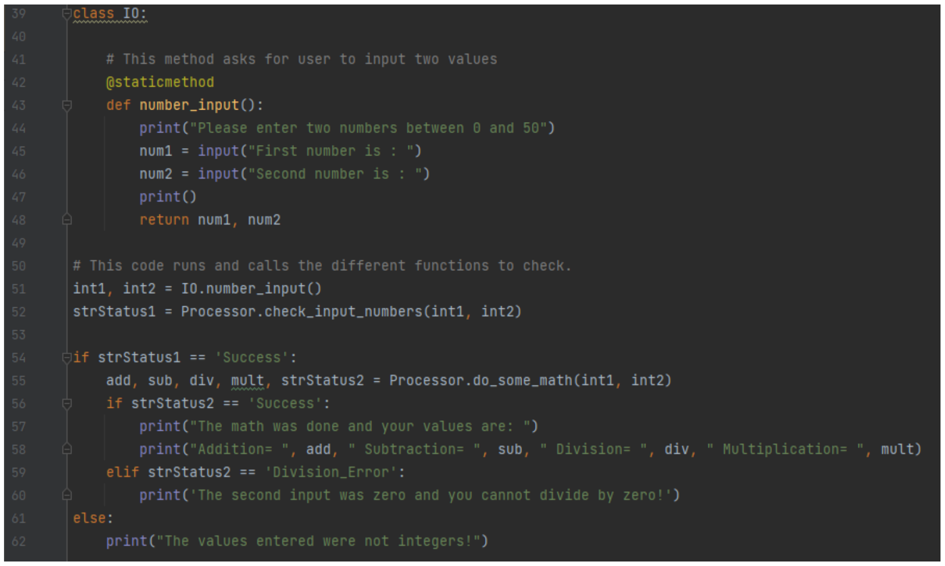
```
Please enter an ID: 1
Please enter a Name: Michael
Contents were appended to File!

Now reading contents from binary file!
[[2, 'Michael'], [1, 'Michael']]

Process finished with exit code 0
```

Figure 2. Snippet of code for Assignment 06.



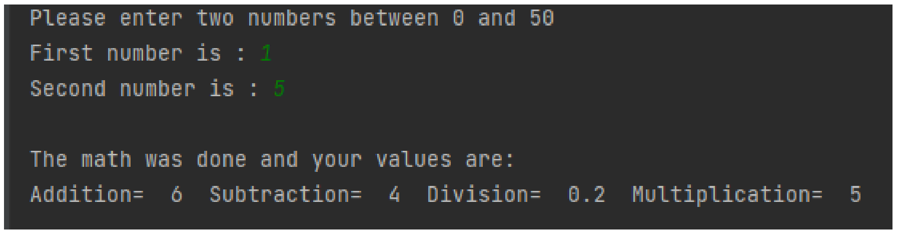
```python
class Processor:

    #  This method checks if the input values were integers.
    @staticmethod
    def check_input_numbers(num1, num2):
        try:
            x = int(num1)
            y = int(num2)
            complex(x,y)
        except ValueError:
            return 'Fail'
        return 'Success'

    # This method does some math.  It returns a division error if the second number is 0.
    @staticmethod
    def do_some_math(num1, num2):
        num1 = int(num1)
        num2 = int(num2)
        try:
            add = num1 + num2
            sub = abs(num1 - num2)
            div = num1 / num2
            mult = num1 * num2
            return add, sub, div, mult, "Success"
        except ZeroDivisionError:
            print("Second value input was zero!")
            print("Cannot divide by zero!")
            return None, None, None, None, "DivisionError"
```

Figure 3. Math processing code.



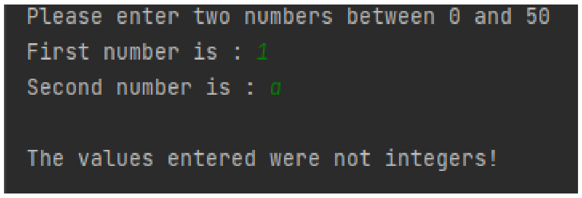
```
class IO:

    # This method asks for user to input two values
    @staticmethod
    def number_input():
        print("Please enter two numbers between 0 and 50")
        num1 = input("First number is : ")
        num2 = input("Second number is : ")
        print()
        return num1, num2

# This code runs and calls the different functions to check.
int1, int2 = IO.number_input()
strStatus1 = Processor.check_input_numbers(int1, int2)

if strStatus1 == 'Success':
    add, sub, div, mult, strStatus2 = Processor.do_some_math(int1, int2)
    if strStatus2 == 'Success':
        print("The math was done and your values are: ")
        print("Addition= ", add, " Subtraction= ", sub, " Division= ", div, " Multiplication= ", mult)
    elif strStatus2 == 'Division_Error':
        print('The second input was zero and you cannot divide by zero!')
else:
    print("The values entered were not integers!")
```

Figure 4. Main part of math script.



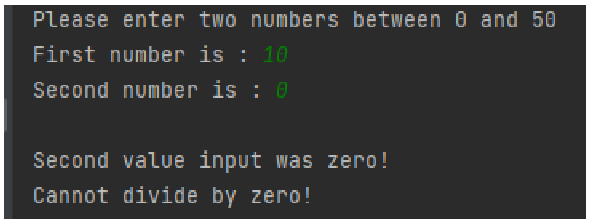
```
Please enter two numbers between 0 and 50
First number is : 1
Second number is : 5

The math was done and your values are:
Addition=  6  Subtraction=  4  Division=  0.2  Multiplication=  5
```

Figure 5. Math script successful run.



```
Please enter two numbers between 0 and 50
First number is : 1
Second number is : a

The values entered were not integers!
```

Figure 6. Math script result of not inputting an integer.



```
Please enter two numbers between 0 and 50
First number is : 10
Second number is : 0

Second value input was zero!
Cannot divide by zero!
```

Figure 7. Math script result when division by zero happens.

Summary

Assignment 7 introduced us to using pickling for binary data and how to handle exceptions. Pickling works much like working with a text file however it is in binary format. Using exceptions in your code is a good way to deal with wrong inputs so that the program doesn't crash.