Daniel Esporma

November 30th, 2021

IT FDN 110 A Foundations Of Programming: Python

Assignment07

<https://github.com/desporma/IntroToProg-Python-Mod07>

Pickling and Structured Error Handling

# Introduction

In this assignment, pickling and structured error handling are discussed. Pickling enables users to store data in a binary format. Structured error handling enables authors to catch and respond to errors that may arise during use of their programs.

# Methodology

## Pickling

Pickling is an operation used to serialize and de-serialize Python objects; writing a Python object to binary data and loading an object from binary data, respectively. Unlike a text file, the binary file is not human readable. The binary format is, however, an effective format for storing and transferring complex data among users.

The Data section of the “Pickling test.py” script begins by declaring variables of multiple data types including a string, an integer, a list, a dictionary, and a table variable “pickle\_all” (Figure 1a). A file is then declared for storing binary data. Though this file “pickle.txt” is not a binary file, it is selected to illustrate how binary data appears in a texfile (Figure 1c). “pickle.txt” will also suffice as storage of binary data for unpickling later.

In the Processing section of “Pickling test.py”, “pickle.txt” is opened and the mode of the open() function is “ab” for the appending of binary data. When data is read from the file, the opposite argument “rb” for the reading of binary data is made.

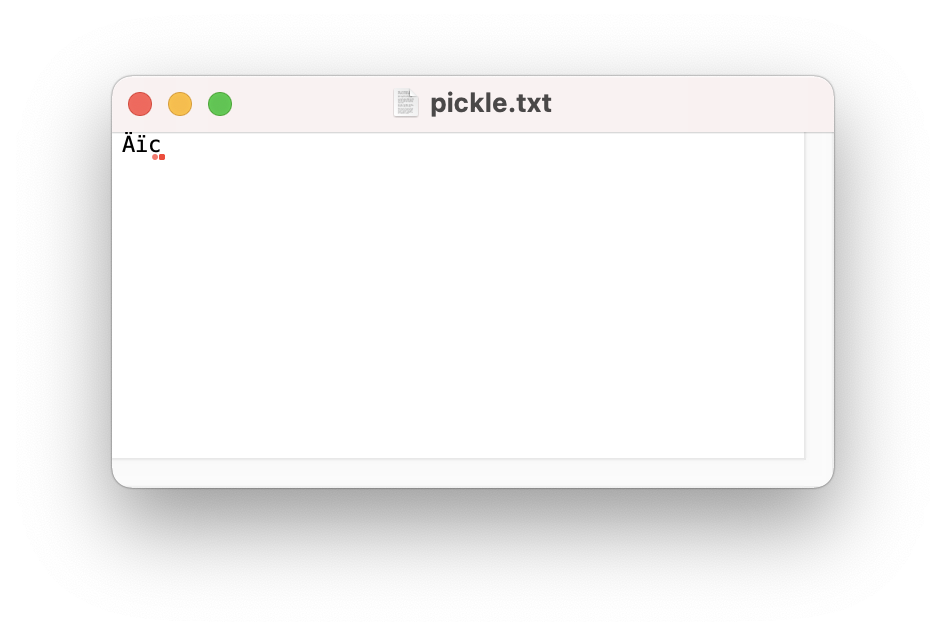
Finally, in the Presentation section of the “Pickling test.py” script prints the Python object de-serialized from pickle.txt (Figure 1b). This data matches “pickle\_all”.

# Data -------------------------------------------- #  
# Declare variables for pickling  
pickle\_str = "hello"  
pickle\_int = 3  
pickle\_lst = ["apple", "peach", "banana"]  
pickle\_dic = {"veg1" : "potato", "veg2" : "broccoli", "veg3" : "spinach"}  
pickle\_all = [pickle\_str, pickle\_int, pickle\_lst, pickle\_dic]  
  
# Declare file  
file = "pickle.txt"  
  
  
# Processing -------------------------------------- #  
# Write pickled representation of pickle\_all as binary code to pickle.dat  
objFile = open(file, "ab")  
pickle.dump(pickle\_all, objFile)  
objFile.close()  
  
# Read the pickled representation (binary) of pickle\_all from pickle.dat  
objFile = open(file, "rb")  
objFileData = pickle.load(objFile)  
objFile.close()  
  
  
# Presentation ------------------------------------ #  
# Print unpickled contents of pickle.dat  
print("Data was unpickled as the following:")  
print(objFileData)

**Figure 1a.** Python code from “Pickling test.py” for pickling multiple types of data.



**Figure 1b.** “Pickling test.py” when run in a terminal.

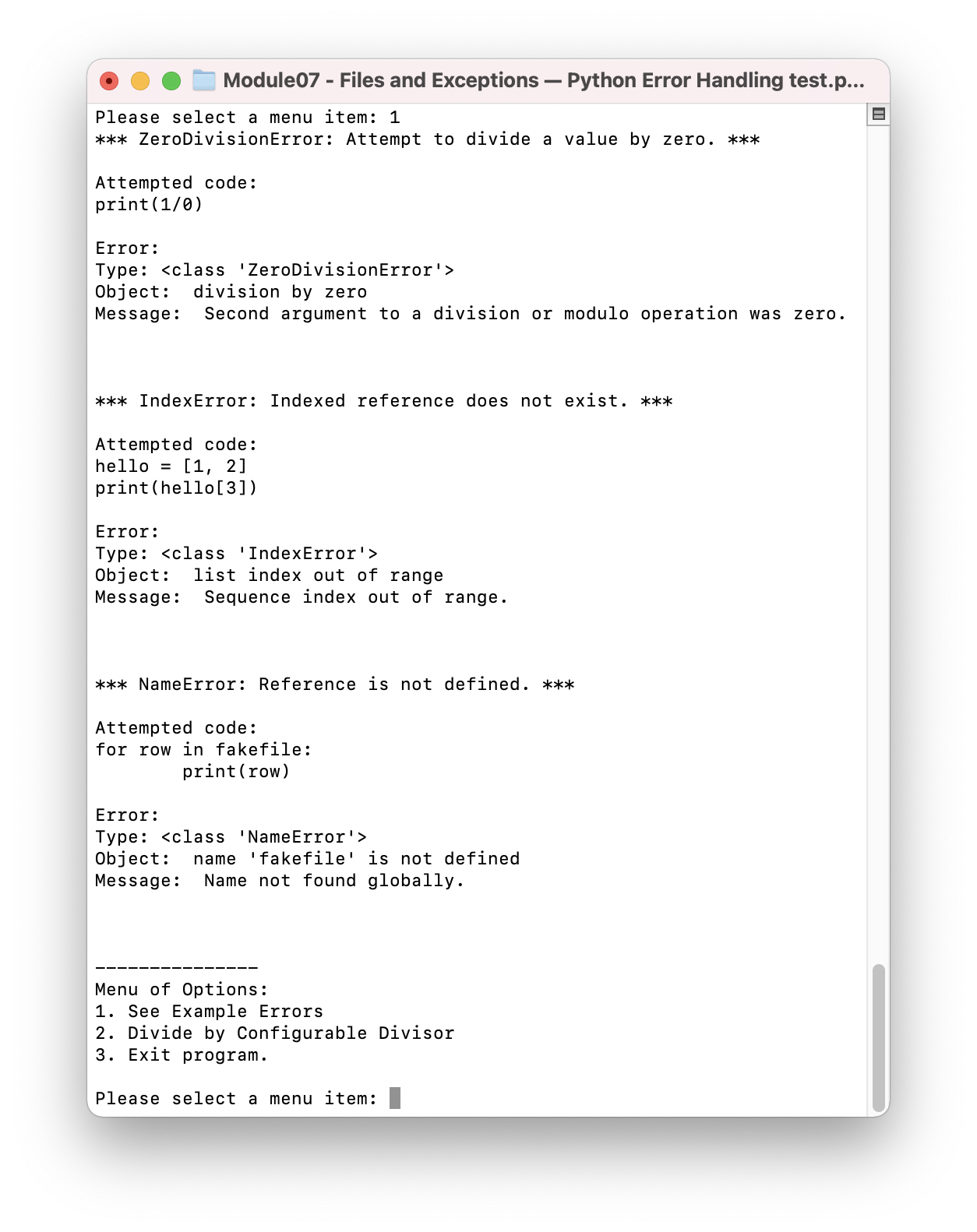


**Figure 1c.** The binary output of pickling using “Pickling test.py”.

## Structured Error Handling

Structured error handling enables authors to catch and respond to errors that may arise when using their scripts or programs. Error classes can be employed, created, or customized to address a variety of potential errors. By using exceptions, authors can even enable their programs to continue running after encountering an error. “Error Handling test.py” is a script written to display a variety of errors to a user and give a user an interactive option to test for errors.

The first option a user may select will showcase various errors (Figure 2a). The error classes shown are a ZeroDivisionError, an IndexError, and a NameError. Each is accompanied with example code to trigger the error, the object type, the error object, and Python code that describes the error class.



**Figure 2a.** Examples of errors printed to the user when running “Error Handling test.py” in a terminal.

The example code in Figure 2a is actually the exact code used in the script (Figure 2b) to generate the data for the Type, Object, and Message fields printed in the terminal view. For example, in the ZeroDiv() function, “print(1/0)” is used to generate an error. An exception is generated and the object is defined as “e”. This object is then entered into the errorcodemsg() function where the type (“str(type(e))”), object (“e”), and message (“e.\_\_doc\_\_”) print commands generate lines found in Figure 2a. This is repeated (with different attempted code for different error classes) for the IndexError and NameError classes in Figure 2a using the IndexErr() and NameErr() functions in Figure 2b. The Presentation section of the script shown in Figure 2c calls all the functions used to generate the examples shown to the user.

# Define functions for error code display.  
class ErrorDisp:  
 @staticmethod  
 def errorcodemsg(e):  
 *"""Standard format in this script for error readout when showing examples."""* ErrorDisp.spacer(0)  
 print("Error:")  
 print("Type: " + str(type(e)))  
 print("Object: ",e)  
 print("Message: ",e.\_\_doc\_\_)  
  
 @staticmethod  
 def ZeroDiv():  
 *"""Zero Division Error example to show user"""* print("\*\*\* ZeroDivisionError: Attempt to divide a value by zero. \*\*\*")  
 ErrorDisp.spacer(0)  
 try:  
 print(1 / 0)  
 except Exception as e:  
 print("Attempted code:"  
 "\nprint(1/0)")  
 ErrorDisp.errorcodemsg(e)  
 ErrorDisp.spacer(2)  
  
 @staticmethod  
 def IndexErr():  
 *"""Index Error example to show user"""* print("\*\*\* IndexError: Indexed reference does not exist. \*\*\*")  
 ErrorDisp.spacer(0)  
 try:  
 hello = [1, 2]  
 print(hello[3])  
 except Exception as e:  
 print("Attempted code: \n"  
 "hello = [1, 2]\n"  
 "print(hello[3])"  
 )  
 ErrorDisp.errorcodemsg(e)  
 ErrorDisp.spacer(2)  
  
 @staticmethod  
 def NameErr():  
 *"""NameError to show user"""* print("\*\*\* NameError: Reference is not defined. \*\*\*")  
 ErrorDisp.spacer(0)  
 try:  
 for row in fakefile:  
 print(row)  
 except Exception as e:  
 print("Attempted code: \n"  
 "for row in fakefile:\n"  
 "\tprint(row)")  
 ErrorDisp.errorcodemsg(e)  
 ErrorDisp.spacer(2)

**Figure 2b.** Examples of errors written as functions called by the Presentation section of “Error Handling test.py”.

# Presentation ------------------------------------ #  
while (True):  
 userdata = str(Menu.userselect()).strip()  
 menurepeat = True  
 if userdata == "1": # Show user examples of errors  
 # ZeroDivisionError  
 ErrorDisp.ZeroDiv()  
  
 # IndexError  
 ErrorDisp.IndexErr()  
  
 # NameError  
 ErrorDisp.NameErr()  
  
 elif userdata == "2": # Give user interactive option to test error handling.  
 Menu.simplediv()  
  
 elif userdata == "3": # Exit program  
 print("Program successfully exited.")  
 break  
 else:  
 print("\nValue must be an integer between 1 and 3. Please try again.\n")  
 menurepeat = False  
  
 continue

**Figure 2c.** The Presentation section of “Error Handling test.py”.

# Define functions for menu  
class Menu:  
 @staticmethod  
 def userselect():  
 print("-"\*15,  
 "\nMenu of Options:"  
 "\n1. See Example Errors"  
 "\n2. Divide by Configurable Divisor"  
 "\n3. Exit program."  
 "\n")  
 selection = int((input("Please select a menu item: ")).strip())  
 return selection

**Figure 2d.** The userselect() function.

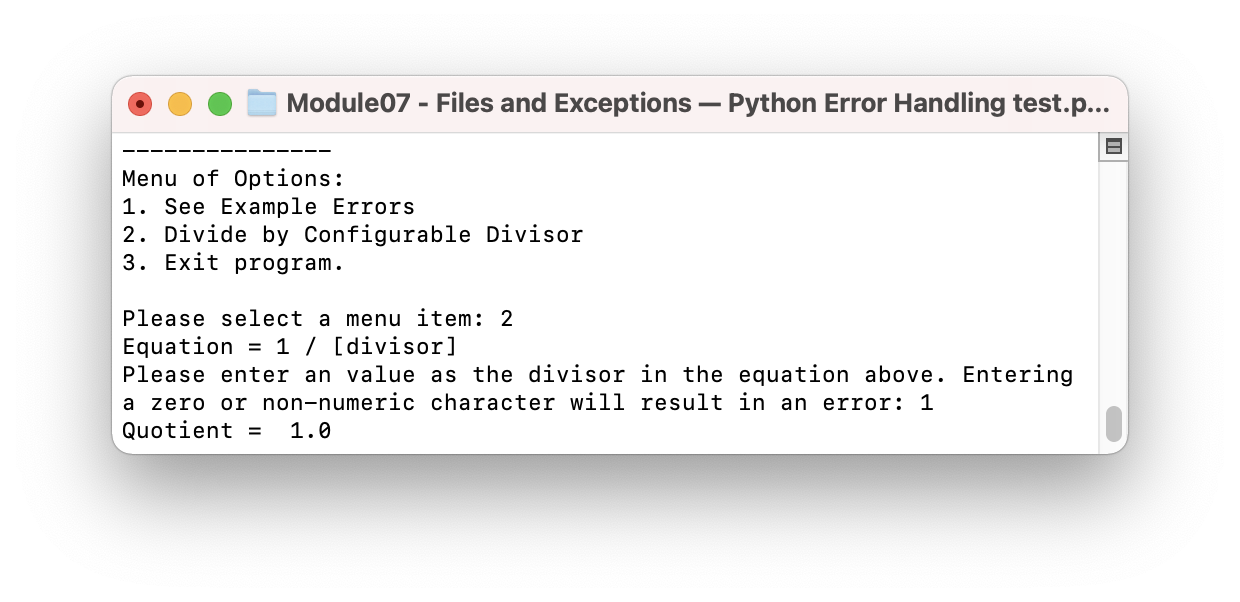
If the user selects the second option when prompted by “Error Handling test.py”, he or she is queried for a divisor to be input into a simple division equation. As seen in Figure 2c and 2d, if a user enters a “2” when prompted by the menu of options, the userdata variable will be defined as “2” and the simplediv() function will be called. As seen in Figure 3a, the simplediv() function uses the input() function to ask the user for a value to define the divisor.

The user is cautioned when prompted by simplediv() that “entering a zero or non-numeric character will result in an error”. As seen in Figure 3b, if a user enters a numeric, non-zero value, a new line appears starting with “Quotient = “ and followed by the quotient of 1 divided by the user’s input value. However, if the user enters a zero or non-numeric value as seen in Figure 3c, a Zero Division Error or Value Error appears respectively.

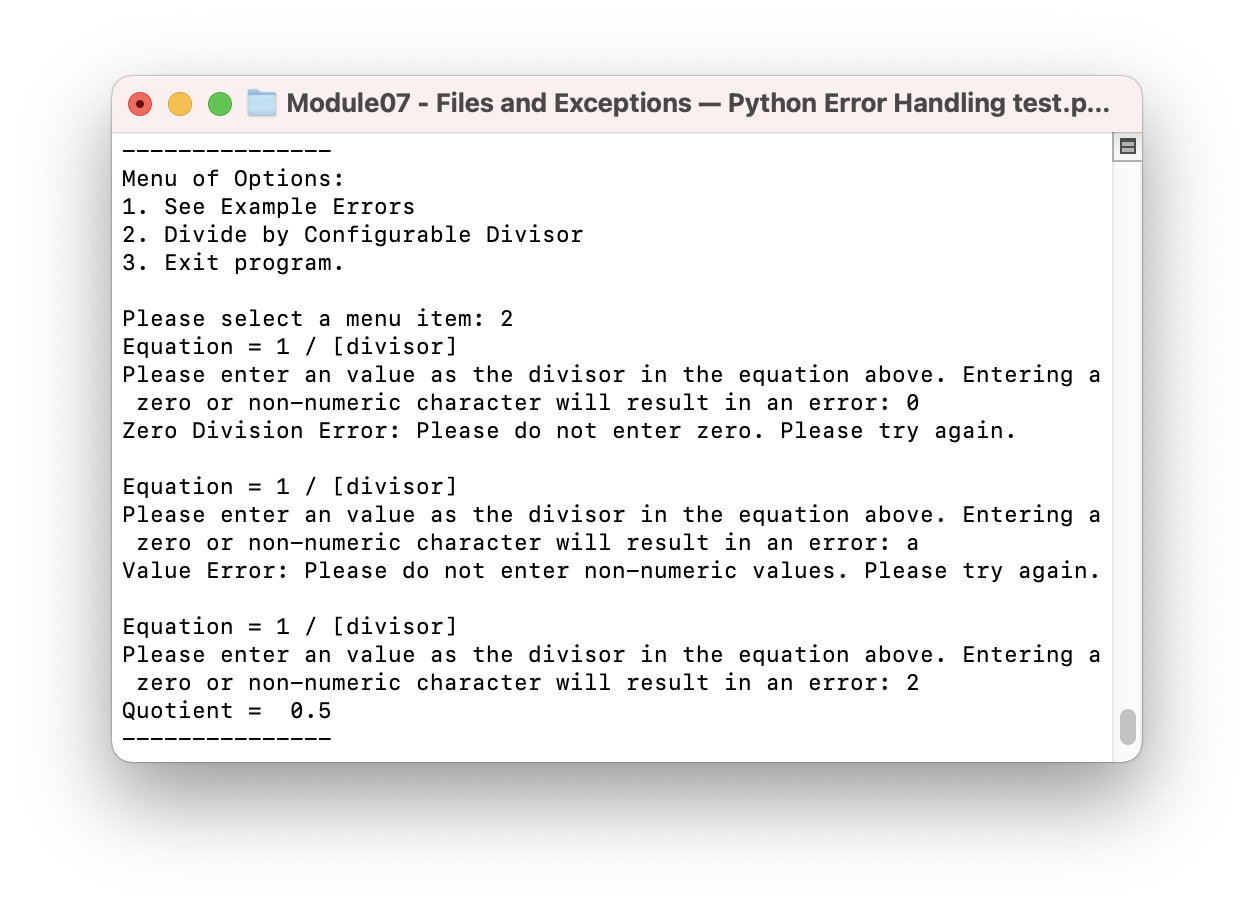
The Zero Division Error and Value Error messages in Figure 3c are specific exceptions caught by the simplediv() function. Each exception also prints a message explaining the error. The function also operates via a while loop, enabling the function to continue until the user inputs an acceptable value.

@staticmethod  
def simplediv():  
 repeat = False  
 while(repeat == False):  
 try:  
 divisor = input("Equation = 1 / [divisor]\n"  
 "Please enter an value as the divisor in the equation above. Entering a zero or non-numeric character will result in an error: ").strip()  
 quotient = 1 / float(divisor)  
 print("Quotient = ", quotient)  
 except ZeroDivisionError as e:  
 print("Zero Division Error: Please do not enter zero. Please try again.\n")  
 except ValueError as e:  
 print("Value Error: Please do not enter non-numeric values. Please try again.\n")  
 else:  
 repeat = True

**Figure 3a.** Error handling built into the simplediv() function of “Error Handling test.py”.



**Figure 3b.** The result of a user entering “1” when prompted by the simplediv() function and viewed in a terminal.



**Figure 3c.** The results of a user entering “0” and “a” when prompted by the simplediv() function and viewed in a terminal.

# Summary

Pickling and structured error handling are valuable skills for writing Python code. Pickling enables object data to be stored in and read from binary data, which enables authors to share data in compact, easy-to-address formats. Should an author wish to import objects from a previous project, importing the pickled version of the data will suffice. Structured error handling enables authors to create robust scripts can respond to specific errors, guide users in their input, and continue to operate after encountering an error.

# External Resources

Below are some helpful resources for more knowledge on pickling and error handling:

<https://docs.python.org/3/library/pickle.html?highlight=pickle#module-pickle>

<https://docs.python.org/3/library/exceptions.html#bltin-exceptions>

<https://docs.python.org/3/tutorial/errors.html>