M.Bruce
Aug 21, 2022
IT FDN 110 (B)
Assignment 07
Link:(GitHub repository)- https://github.com/Jane2024/IntroToProg-Python-Mod07
Link:(GitHub Webpage)- https://jane2024.github.io/IntroToProg-Python-Mod07/

Pickling and Error Handling in Python

Introduction

This week covered reading and writing to text files, reading and writing to binary files using Pickle and error handling with try/except blocks. In addition, learning how to use markdown language to format a GitHub web page. The assignment for the week had us research Pickling in Python and error handling, then we needed to create demos to shown these principles in action. This knowledge document will cover the above topics and also be reflected on the newly created GitHub web page.

Text files

Read and Write modes

In Python, we have learned to open a file for read or write:
file = open(file_name, "r"), file = open(file_name, "w"), then
used ***data= file.read()*** and ***file.write(strData +"\n")*** file objects to read and write data from and to text files.
Then closed the file: file.close().

This would read in all data from a text file or write a string to a text file. There are other options that extend this functionality to perform more specific read/write tasks.

Append Mode

When opening a file for write in "w" access mode, if there is existing data in a current file, it will be deleted. If opening a file for write, numerous write calls can be done and it will append data as long as the file remains open. Once the file is closed, a new open for write call in "w" access mode will delete the contents and overwrite with new content. Additional access modes allow new data to be appended to existing files. One would be with the ***append*** access mode: ***open(file_name, "a").***

Other reading options

Other read-based access modes include:
open(file_name, "r+") Read and write from text file. If the file does not exist Python will throw an error.
open(file_name, "a+") Append and Read from text file. If the file exists, it is appended to, if it does not exist it will be created
open(file_name, "w+") *Write to and read* from a text file. If the file exists, it will be overwritten, if it does not exist it will be created.

When a call is made to read a file using the ***open(file_name, "r")*** followed by ***data = file.read() object method***, the entire contents of the file is returned as string and assigned to the variable. But sometimes the desired action is only to read a specific amount of data like a character or line of data. This can be done by modifying the read object method and placing a number between the parentheses.

For example: ***data = file.read(5)*** will read the first 5 characters in the file. The next read call (while the file remains

open) will start up at the sixth character. This can continue till the end of the file, at which point empty strings will be returned. To read from the beginning of the file, then closing and reopening for read will restart at the beginning of the file.

The ***readline()*** method allows reading the current line of data from the file or specifying a number of characters from the current line to be read. Each new call to readline() (as long as the file is open) will advance and read the next line of data. Advancing one line at a time is called cursor. Closing and reopening the file starts reads at the first line again. A "while" loop can be used to advance through a file reading multiple lines. ***Listing 7.1*** below shows a function that handles reading multiple rows using the readline() method.

```
def read_some_data_rows(file_name, number_of_rows):
    """ Reads rows of string data from a file

    :param file_name: (string) with name of file
    :param number_of_rows: (int) with number of rows you want from the file
    :return: (list) with one or more data rows read from the file
    """
    counter = 0
    data = []
    file = open(file_name, "r")
    while counter < number_of_rows:
        data.append([file.readline()])  # APPENDING the data to a list
        counter += 1
    file.close()
    return data  # returning the chosen row of data
```

LISTING 7.1 code segment for function that reads multiple lines from a file (while loop)

The ***readlines()*** method, reads all the lines in a file, and returns ***a list***. This is different than the read() function, which reads all the lines in a file and returns ***a string***. An example of reading a file with existing rows and the readlines() method is listed below in ***Listing 7.2***.

```
def read_file_data_to_list(file_name):
    """ Reads rows of data data from a file into a list

    :param file_name: (string) with name of file
    :return: (list) with rows of data read from the file
    """
    file = open(file_name, "r")
    data = file.readlines()  # reads rows of data into a list object
    file.close()
    return data

# Presentation ------------------------------------ #
print('Here is the first row of data from the file:')
lstData = read_file_data_to_list(file_name=strFileName)
print(lstData[0].strip())  # used strip() to remove newline
print(lstData[1].strip())
```

LISTING 7.2 code segment for function that reads multiple lines from a file (readlines())

Another option to **read multiple rows** of data would be to use a "for" loop. A ***for*** loop automatically closes the file when it reaches the end of the file's data.

Lastly, the **"with"** construct, which helps make the code cleaner and easier to read and also automatically closes the file when the code reaches the end of the "with" block. This ensures that a resource is "cleaned up" when the code using it finishes running, regardless if exceptions are thrown. Example code block is in ***Listing 7.3*** below.

```
with open("text_file.txt", "w") as file:
     file.write("hello World!" + "\n")
```

LISTING 7.3 **code segment for function using "with" construct**

Syntactic sugar is a syntax that allow humans to write code easier, it makes the code sweeter. The **"with"** construct is an example.

Binary files

In Python, when data is serialized (converted to a byte stream) and stored in a binary file, it is called **"pickling".** When it is deserialized (converted back to a Python object), it is called un-pickling. Any type of python object can be pickled.

Lab 7-1 Working with binary files

For Lab 7-1, create a function that saves data to a binary file. Then open the file and read its contents. See **Figure 7.1** for Lab7-1 code that defines functions that save and read binary data

```
# ------------------------------------------------- #
# Title: Lab7-1
# Description: A simple example of storing data in a binary file
# ChangeLog: (Who, When, What)
# MBruce,8.21.2022,Created Script
# ------------------------------------------------- #
import pickle  # This imports code from another code file!

# Data --------------------------------------------- #
strFileName = 'AppData.dat'
lstUsers = []

# Processing --------------------------------------- #
def save_data_to_file(file_name, list_of_users):
    # TODO: Add code here
    file = open(file_name, "wb")
    pickle.dump(list_of_users, file)
    file.close()

def read_data_from_file(file_name):
    # TODO: Add code here
    file = open(file_name, "rb")
    list_of_users = pickle.load(file)
    file.close()
    return list_of_users
# Presentation ------------------------------------- #
# TODO: Get ID and NAME From user, then store it in a list object

intID = int(input("Enter an Id: "))
strName = str(input("Enter a Name: "))
lstUsers = [intID, strName]

# TODO: store the list object into a binary file
save_data_to_file(strFileName, lstUsers)

# TODO: Read the data from the file into a new list object and display the contents
print(read_data_from_file(strFileName))
```

FIGURE 7.1 **Lab7-1 script code for functions to save**

See **Figure 7-2 below** for Lab7-1 output when run in cmd shell. The program prompts for user to input an ID and a Name. It stores the ID and name in variables and adds those to a list of users. A function is called to open a file and store the list of users in binary form(pickle it). Then a function is called that reads the binary data from the file (un-pickles it) and returns it in list form. That call is made within a print function, so the read function is immediately evaluated and printed out.

```
C:\Users\MBruce>cd c:\_PythonClass\Assignment07

c:\_PythonClass\Assignment07>python.exe C:\_PythonClass\Assignment07\Lab7-1_Starter.py
Enter an Id: 07
Enter a Name: Fitz
[7, 'Fitz']

c:\_PythonClass\Assignment07>
```

FIGURE 7.2 Lab7-1 output

Error Handling

Try-except blocks in Python allow errors to be caught and handled in a customized way. Built-in error handling can be cryptic to understand and customizing a response in the try-except block can provide more useful information when an error occurs. In addition, Python has an "exception" class that can create exception objects to hold data about the error when one occurs. In this class are built-in classes to handle specific errors. The try statement always runs and contains the portion of the code that is being check for error. If an error occurs, the except clause handles it. If no error occurs, the except clause does not run, the program bypasses it. The exception object is captured in the "except" section of the try-except block. See ***Listing 7.4*** below.

```
try:
    quotient = 5/0
    print(quotient)
except Exception as e:
    print("There was an error! << Custom Message")

    print("Built-In Pythons error info: ")
    print(e)
    print(type(e))
    print(e.__doc__)
    print(e.__str__())
```

LISTING 7.4 code example of a try-except block which captures the exception in "e". Then uses that in print statements to provide the built-in error reporting.

What each statement prints out:
print(e)- basic name of error "division by zero"
print(type(e))- the exception class that handled the error "<class "ZeroDivisionError">
print(e.__doc__)- the doc string error message "The Second argument to a division or modulo operation was zero".
print(e.__str__())- the error string "division by zero"

GitHub Pages

A GitHub page uses “markdown” language to format information in the page. It is converted into HTML via an application called Jekyll. A “index.md” file contains the markdown commands and data for the GitHub page. This files is located in the main branch in the docs folder in each GitHub repository.

Assignment 07

This assignment is to create a demo script that provides an example of the use of Pickling in Python and Error Handling.

Create Script

1) Opened PyCharm and created folder named “Assignment07”
2) Created a new project **Assignment07.py** at:
   C:/_PythonClass/Assignment07/ **Assignment07.py**

Example script demoing pickling, un-pickling and error handling

I decided to do a combination script which would use pickling to store a list of numbers in a binary format, then use un-pickling to load the list from file and error handling to check that the file existed and if not inform the user.

Pickling

A function “create_file()” was designed to create a list of numbers from 1 to 25, store them in a list and then store that list in a binary file named “Numbers.dat” using “pickle.dump”. Figure 7.3 shows the code for that function below.

```
#############     Import Modules        #################
import pickle


#############     Define Functions      #################

def create_file():
    num_list = []                           # Define list to hold numbers
    for k in range(25):                     # Loop to create list of numbers 1 to 25
        num_list.append(k + 1)              # append each number to list

    file_out = open("Numbers.dat", 'wb')    # Open file to write binary data
    pickle.dump(num_list, file_out)         # Employ pickle to dump the data into the file
    file_out.close()
```

FIGURE 7.3 Assignment7 code utilizing pickle.dump()

Un-Pickle

Another function was designed to unpickle the file into a list and allow the user to search that list for a number. Variables were created to hold the file name “strFilename”, a Boolean flag to be set if the number was found “bolFoundIt”, a list for the numbers “list_of_numbers”.
Then “pickle.load(file)” was used to de-serialize the data in the binary file “Numbers.dat” and store in the list variable- “list_of_numbers”.

A For Loop was used to cycle through the list to check for the user provided number. If found, it notifies the user it found the number in the list, sets the flag "bolFoundIt" to true and exits. Else it will continue the loop till it finds the number or reaches the end of the list. Once the end of the list is reached, it exists the for loop and a check is done of the flag to determine if the number was found. If not, it notifies the user and exits program. See code for this function in ***Figure 7.4*** below.

```
def num_search():
    strFilename = 'Numbers.dat'                          # Assign the binary data file to variable
    bolFoundIt = False                                   # boolean to be set if number is found
    list_of_numbers = []                                                  # List for numbers
    int_FindNum = int(input("Enter number to search for between 1 and 25: "))   # prompt user for number

    try:                                                  # Start of Try/except error handling block to check if file is found
        file = open(strFilename,'rb')                     # Opens the file for reading binary data
        list_of_numbers = pickle.load(file)               # unpickles data from file into list

        for x in range(len(list_of_numbers)):             # loop to cycle through in search of user number

            if list_of_numbers[x] == int_FindNum:         # If number is found, print found it!
                print("Found your number in list:", int_FindNum)
                file.close()
                bolFoundIt = True                         # Set boolean flag to true if number is found
                break
            else:                                         # If not found, continue to cycle through the list
                continue
        if bolFoundIt != True:                                        # When for loop exits, if flag is not true, number was not found
            print("Reached end of list and your number was not found.")
            file.close()
    except IOError:                                       # If file was not found, then except clause executes and informs the user.
        print("ERROR...cannot locate", strFilename, "file.")
```

FIGURE 7.4 Assignment7 code utilizing pickle.load()

The body of the program simply calls the two functions to create and file and store it, then load it and search it. ***Figure 7.5*** shows this below.

```
##############       Main          ############################
# create a binary file of a list of numbers using pickle
create_file()

# load the file and let user search for a number
num_search()
```

FIGURE 7.5 Assignment7 code main body

When the program is run, and the user enters a number, if it is found, the user is notified – Figure 7.6

```
C:\Users\MBruce>cd c:\_PythonClass\Assignment07

c:\_PythonClass\Assignment07>python.exe C:\_PythonClass\Assignment07\Assignment07.py
Enter number to search for between 1 and 25: 5
Found your number in list: 5
```

FIGURE 7.6 Assignment7 output- successfully found number

If the user entered number is not found, then the user is notified as well Figure 7.7

```
c:\_PythonClass\Assignment07>python.exe C:\_PythonClass\Assignment07\Assignment07.py
Enter number to search for between 1 and 25: 59
Reached end of list and your number was not found.
```

FIGURE 7.7 Assignment7 output- number not found

Error Handling

A try/except block is placed around the code segment which loads the file into the list. If the Numbers.dat is not found in the current working directory as the python script, then the except clause will execute and inform the user the file could not be located and exit the program. A test was done without running the "create_file()" function to test if the except clause would catch the error. It did. See Figure 7.8.

```
c:\_PythonClass\Assignment07>python.exe C:\_PythonClass\Assignment07\Assignment07.py
Enter number to search for between 1 and 25: 5
ERROR...cannot locate Numbers.dat file.
```

FIGURE 7.8 Assignment7 Exception caught and user notified of error for file not found.

Summary

In the module07, the main take away for me was the use of pickling to serialize data and store in a binary file using pickle.dump() and then how to de-serialize it, by unpickling the file using pickle.load(). Not sure how much I will use it, since research showed it is not secure and is dependent on both ends of the process utilizing the same version of python. But it is a handy thing to learn as a newbie to Python and an alternate to storing to a text file. The TRY/BLOCK error handling is something that will be used in every program I create going forward since it is critical to allow for a program to catch errors and fail gracefully. There is an extensive the list of built-in exception errors to help handle this, or I can build custom ones. The last task was to take the knowledge document and format it for display on my newly created GitHub web page using markdown language.