IMarshall

02/27/2023

IT FDN 110 A

Assignment 07

Github Repository: IMarshallUW/IntroToProg-Python-Mod07: Files and Exceptions (github.com)

Github Webpage: IntroToProg-Python-Mod07 | Files and Exceptions (imarshalluw.github.io)

Assignment 07 – Files and Exceptions

Introduction

This module we were challenged to write our code in a instructional format to simulate introducing pickling and exception handling to someone unfamiliar to the concepts, and then wrapping both those concepts together using functions. We chose to separate the code into three separate file so they could be built on to introduce new concepts and make them more of a finished product as we went. In this paper we will go over each of those files individually.

File: Assignment07_01 Pickling

To begin, after the title block and declaring our variables we begin each file with a description of the topic we'll be covering in the hopes of giving the student some background on what they'll be learning in the script and maybe why it would be used. The first topic to cover was pickling.

Pickling is a function that converts any Python object (list, dict, etc.) to and from binary. This allows for easier transfer between databases to then be stored in a file or database. We started with the import pickle command which lets the program know that at some point in the script it will importing binary code from another file. We then used a very basic code to obtain a customer name and phone number and add them to a list in memory. Followed by the use of the pickle.dump and how it works, explaining that it still writes the code to a file, just an extra bit of code to tell the program to write it as binary. To close out the program we use the pickle.load command to read the code in the file and show it to the user, but highlight that that command will only show a single row (figure 01). In the next file we will show how to make the command more complex to read the whole file.

```python
import pickle  # Imports code from another file

strname = str(input('Please enter a first and last name: ').strip())
intPhone = int(input('Please enter a phone number: ').strip())
lstcustomer = [strname, intPhone]
'''print(lstcustomer)'''
# Duplicated print to user with print(objFileData below, removed to clean visual

# We will use pickle.dump to store the data, it works the same as writing the data to a file, just lets the system
# know it's binary.
objfile = open('CustomerContact.txt', 'ab')
# 'ab' appends the data in the file, allowing us to add to it without overriding it while defining it as binary.
pickle.dump(lstcustomer, objfile)
objfile.close()

# To verify the data saved we will read back the data contained in it with pickle.load
objFile = open("CustomerContact.txt", "rb")
objFileData = pickle.load(objFile) #load() only loads one row of data.
objFile.close()

print(objFileData)
```

Figure 1: Assignment07_01 Pickling script

File: Assignment07_02 Exception Handling

In the next file we built on our first bit of code to add some minor complexities and introduce the concept of exception handling. We started with the same style of introduction block as before to explain that exception handling is the use of a try/catch block that will return the user back to a certain point in the program if an exception is encountered instead of breaking and ending the program.

To start building on our code we showed the student how to use a while loop to help ensure that the user enters a first *and* last name as instructed by checking for a space in their entry (figure 2). We then proceeded to discuss how in the previous program, there was no enforcement or check on if the phone number that was entered was a complete ten digit phone number and that if letter or other non-number value was entered the program would break (figure 3). So we showed the student how to use a while loop to check the character length to ensure that the phone was long enough to be a full seven digit phone number with a three digit area code. Then once the length was correct we used a try block to convert the string into an integer. Whereas before the program would break and kick the user out if the entry contained anything that wasn't a number, now the user is presented with a error message by our own design as well as the native Python error message and sent back to the beginning of the while loop to try again (figure 4).

```python
while True:
    strname = str(input('Please enter a first and last name: ').strip())
    # A simple way to help mitigate people only entering their first name is to have the program check to see
    # that the entry contains a space.
    if ' ' not in strname:
        print('Invalid, please ensure you are entering your first AND last name.')
    else:
        break
```

Figure 2: Assignment07_02, validating user entered first AND last name

```
C:\_PythonClass\Assignment07>Python Assignment07_01.py
Please enter a first and last name: Steve
Please enter a phone number: Cooper
Traceback (most recent call last):
  File "C:\_PythonClass\Assignment07\Assignment07_01.py", line 22, in <module>
    intPhone = int(input('Please enter a phone number: ').strip())
ValueError: invalid literal for int() with base 10: 'Cooper'
```

Figure 3: Assignment 07_02, program breaks if non-integer is entered

```
while True:
    strPhone = str(input('Please enter a domestic phone number with area code (numbers only): ').strip())
    if len(strPhone) != 10:
# In the last program the user could enter any number they wanted
# and if anything was entered that wasn't a number the program would break. To prevent this we did the following:
#   Checks to make sure that there are 10 characters entered
#   Changed to start as string and check for length first, making sure user enters full phone number length
#   Then become integer and check to see if data are numbers.
#   since data entry didn't recognize 0 as an integer if user entered it intitially
        print('Invalid phone number length. Please enter a 10 digit phone number.')
    else:
        try:
            intPhone = int(strPhone)
            break
        except ValueError as ve:
            print('Your input must be a number.')
            print('Native Python error message: ')
            print(ve, ve.__doc__, type(ve), sep='\n')
        except Exception as e:
            print('Please validate your entry and try again.')
            print('Native Python error message: ')
            print(e, e.__doc__, type(e), sep='\n')
# In Assignment 07 Pickling if the user entered any values that weren't integers the program would break.
# This try statement takes the same input from before, but now if an error occurs where the user
# enters a non-integer value we prompt them to only enter number, any other exceptions will prompt the user to
# validate their entry. Allowing them to try again without being kicked out of the program.
lstcustomer = [strname, intPhone]
```

Figure 4: Assignment07_02, check for phone number length and exception handling to prevent program failure if non-integer entered

The last part of the code that we changed allows the pickle.load function to read all the data in our file instead of just a single line. We accomplished this by showing the student that the pickle.load function could be combined with a while loop combined with a try/catch block. The try statement tells the program to read and print the first line in the file with no break statement on success. This make it repeat the loop and read the next line. It will continue this until there is no more data and it receiving a EOFError, or end of data error. Which will then break the loop (figure 5).

```
# Use a while loop to continually run the pick.load function until there was no more data to load.
# Otherwise known as a EOFError.
with open('CustomerContact.txt', 'rb') as objfile:
    while True:
        try:
            lstcustFile = pickle.load(objfile)
            print(lstcustFile)
        except EOFError:
            break
```

Figure 5: Assignment07_02, while loop to use pickle.load for all lines in file

File: Assignment07_03 Using Funcitons

The final file took the expanded code we had just written and showed how we could do the same thing using functions. We started Assignment07_03 by splitting our data entry into two distinct input/output functions for the customer name and phone number, and a separate save function. We added an .upper() the name entry for conformity of the data, and a return statement at the end of our input/output functions so the data entered became variables to be used by our save function. We also added a print statement to show the user what they had just entered. Other than those small adjustments the code is identical to what we wrote in Assignment07_02 (figure 6 & 7).

```
def get_name():
    while True:
        strname = str(input('Please enter a first and last name: ').strip().upper())
        if ' ' not in strname:
            print('Invalid, please ensure you are entering a first AND last name.')
        else:
            return strname
        # return identifies the data entered as a variable to be used later in another function

def get_phone():
    while True:
        strPhone = str(input('Please enter a domestic phone number with area code (numbers only): ').strip())
        if len(strPhone) != 10:
            print('Invalid phone number length. Please enter a 10 digit phone number.')
        else:
            try:
                intPhone = int(strPhone)
                return intPhone
            except ValueError as ve:
                print('Your input must be a number.')
                print('Native Python error message: ')
                print(ve, ve.__doc__, type(ve), sep='\n')
            except Exception as e:
                print('Please validate your entry and try again.')
                print('Native Python error message: ')
                print(e, e.__doc__, type(e), sep='\n')
```

Figure 6: Assignment07_03, get_name and get_phone functions

```
def save_customer():
    strname = get_name()
    intPhone = get_phone()
    lstcustomer = [strname, intPhone]
    print('-----The following data was written to file-----')
    print(lstcustomer)

    objfile = open('CustomerContact.txt', 'ab')
    pickle.dump(lstcustomer, objfile)
    objfile.close()
```

Figure 7: Assignment07_03, save_customer function

Our script to read the file remained the same entirely other than being defined as a function (figure 8). And finally we defined a menu that would act as the user interface. Which we used an if statement with to show the menu if the program were opened directly from file. Otherwise the menu would not be shown. This allows for other programmers to import our code and utilize our functions without forcing our interface (figure 8). To make it easier for the student to see how the code was changed the code from Assignment07_02 is at the bottom of the script, commented out, with notes on how it was parsed out to make our functions in the hopes of demonstrating that once the script is written it can be converted into functions fairly easily with a plan.

```
# We made a menu interface to allow for easier user interface and multiple data entry.
def menu():  # User interface menu
    while True:
        choice = input('''
       Please select one of the following:
        1) Save a new customer
        2) View customers on file
        3) Exit program
        ''')
        if choice == '1':
            save_customer()
        elif choice == '2':
            read_customers()
        elif choice == '3':
            break
        else:
            print('Invalid choice, please try again.')
    print('Thank you and have a nice day.')

if __name__ == '__main__':
    menu()
    # Checks if program is being executed directly from file. If the program is being
    # imported the user can still use the functions but the menu will not open on starting.
```

Figure 8: Assignment07_03, user menu and if statement

Conclusion

In this paper we showed how used commented out verbiage in a program script to instruct someone on how to write a script to work with pickling for handling binary. We then built on that program to add some constraints and demonstrate exception handling. Finally, we again built on that script to perform all the things that we learned using functions. As always reference below for our three scripts running in Command Prompt (figure 9 & 10).

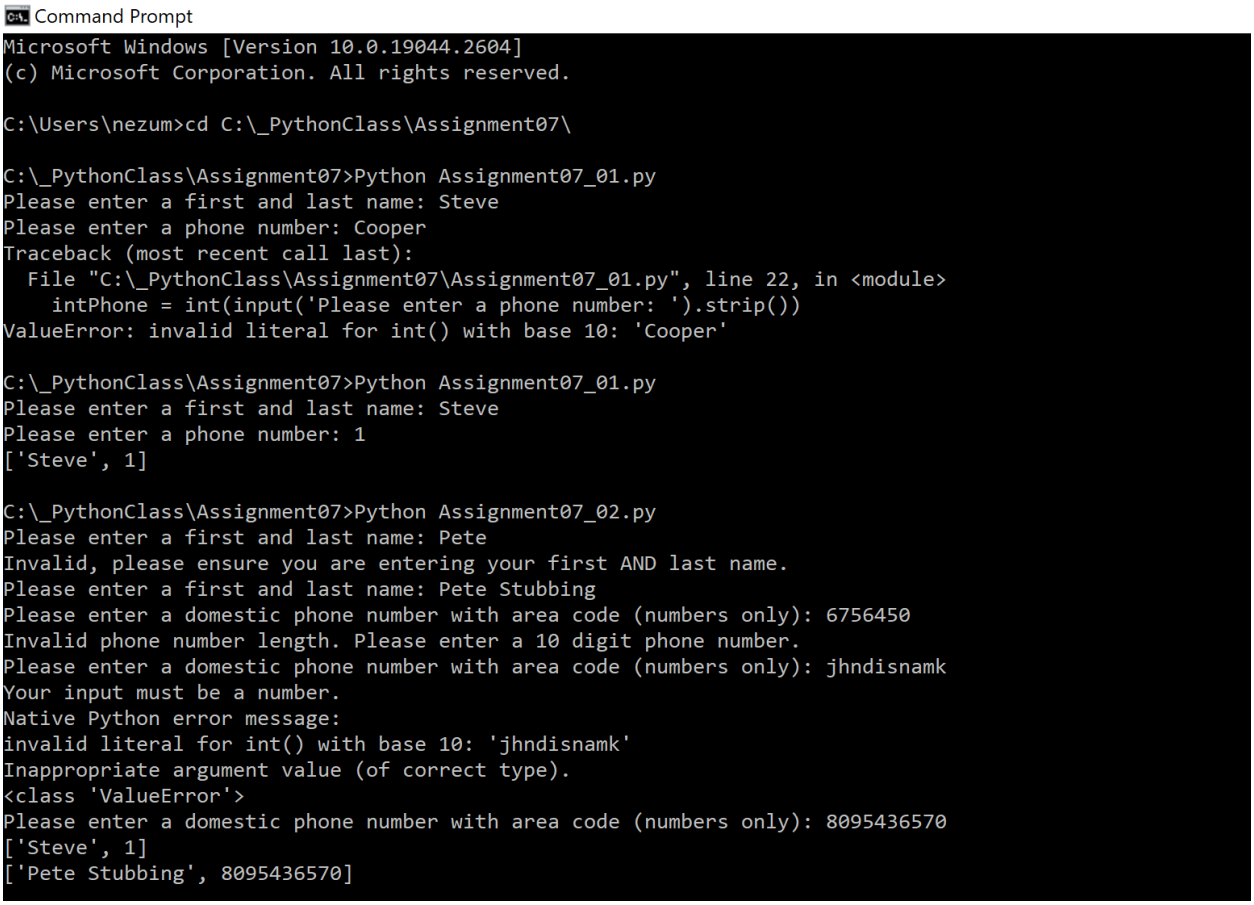

Figure 9: Run in command prompt

```
C:\_PythonClass\Assignment07>Python Assignment07_03.py

       Please select on of the following:
        1) Save a new customer
        2) Read customers on file
        3) Exit program
        1
Please enter a first and last name: Mike Miller
Please enter a domestic phone number with area code (numbers only): 3427360979
-----The following data was written to file-----
['MIKE MILLER', 3427360979]

       Please select on of the following:
        1) Save a new customer
        2) Read customers on file
        3) Exit program
        2
['Steve', 1]
['Pete Stubbing', 8095436570]
['MIKE MILLER', 3427360979]

       Please select on of the following:
        1) Save a new customer
        2) Read customers on file
        3) Exit program
        3
Thank you and have a nice day.

C:\_PythonClass\Assignment07>
```

Figure 10: Run in command prompt cont