Meredith Shapiro

August 25, 2022

IT FDN 110 B - Foundations of Programming: Python

Assignment 06

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

Top of Form

Bottom of Form

Top of Form

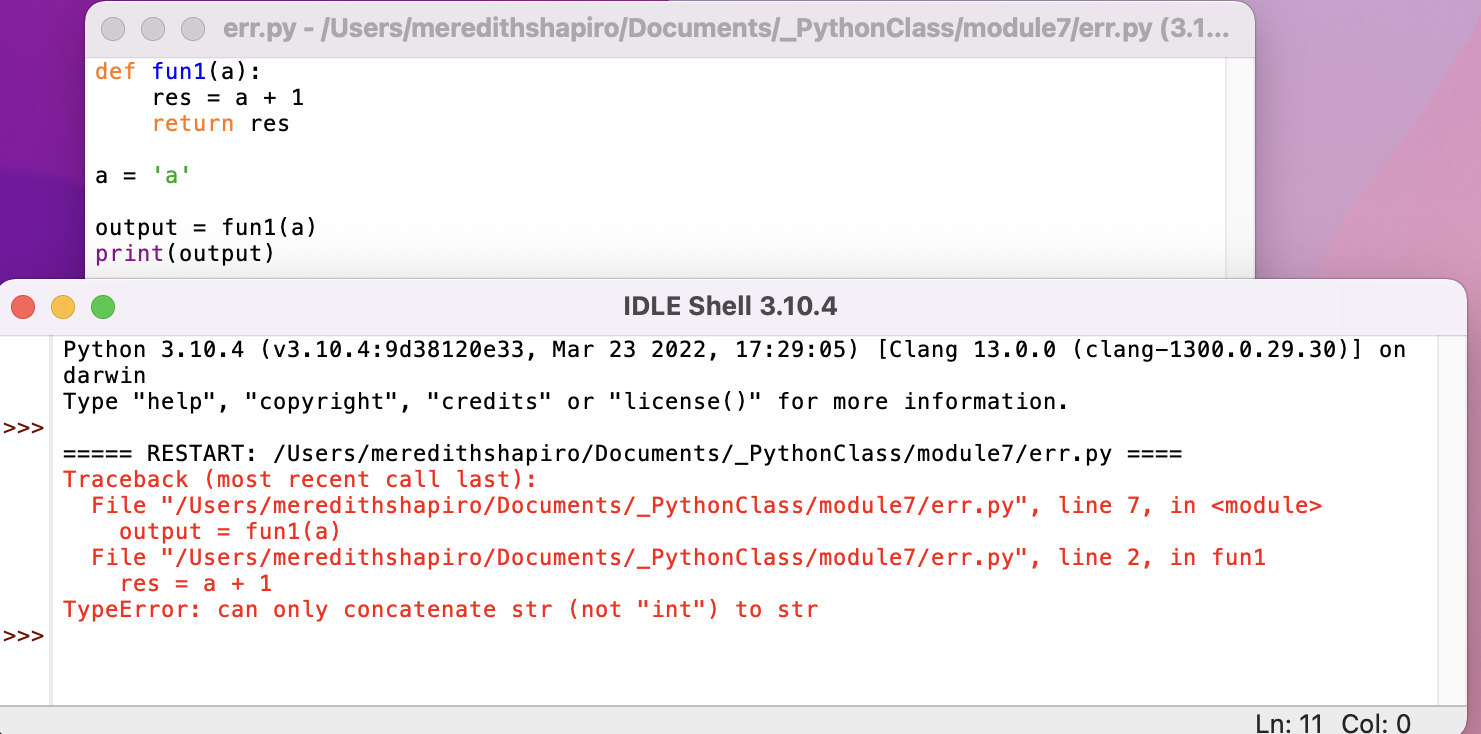
Bottom of Form

**Error Handling and Pickling**

**Introduction**

In this markdown document, I will explain the basics of how to display custom error messages for users of our Python scripts. I will also cover pickling (binary files).

**Error Handling**

When user input is required as part of running a script, this opens the door for user error. Any user can, and most likely will, introduce errors while running a program. As we know from developing code, when you make an error in Python, you will see a system generated error message in red text. But, these messages are sometimes too technical or not informative enough for a user. [](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/error_message.png) **Figure 1. System-generated error message from running a function with incompatible data**

We can instead add a "try-except" block to our code, which will allow the program to capture errors, and display more informative error messages.

To use try-except, all the code you are running (the main body, not the section where you define functions) can be enclosed in "try: "

#--------------------------------------------------------------------------#

# Title: Listing 1

# Description: Excerpt of the "try" portion of my script

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#--------------------------------------------------------------------------#

try:

fltV1 = float(input("Enter value 1: "))

fltV2 = float(input("Enter value 2: "))

twoVals(fltV1, fltV2)

Make sure to indent everything that happens within the "try." After the main body of the script enclosed in "try: ", you will add "except: " to tell Python how to handle exceptions (errors). You could just print a generic error message, like this:

#--------------------------------------------------------------------------#

# Title: Listing 2

# Description: Example of a generic exception with message

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#--------------------------------------------------------------------------#

except Exception as e:

print("There was an error!")

That would solve the issue of the error message being too technical, but it doesn't help it be more informative. We can get the details of an error by using Python's Exception class:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 3

# Description: Code fragment showing how to get additional details on an exception, and its type

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

except Exception as e:

print(e, e.\_\_doc\_\_, type(e), sep='\n')

We can print the exception itself (e, the name we have given it) as well as more detailed documentation, and the type of error - type(e).

Once we have these details, we can create separate Exceptions for each type of error that seems likely, and display custom messages for them. If we know the type of error, we can replace "Exception" in the "except Exception as e:" statement, with whatever the actual exception type is. For instance, I wrote a function that takes input for two numeric values and divides them. I wanted to see what would happen if I entered something non-numeric:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 4

# Description: Code fragment showing how to get additional details on an exception, and its type

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

try:

fltV1 = float(input("Enter value 1: "))

fltV2 = float(input("Enter value 2: "))

twoVals(fltV1, fltV2)

fltV1, fltV2, fltQuot = twoVals(fltV1, fltV2) # unpack tuple

output = displayVals(fltV1, fltV2, fltQuot)

print(output)

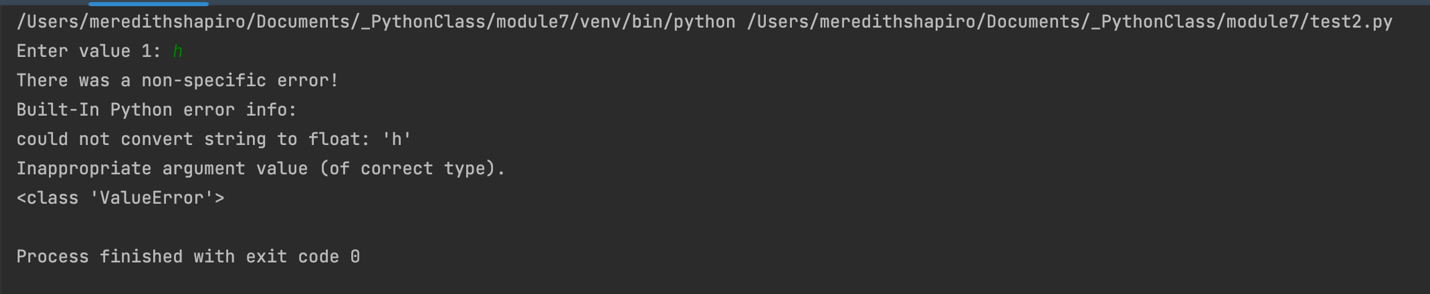
except Exception as e:

print("There was a non-specific error!")

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

I tried running this but, when asked for my input for the first value, I entered "h". You can see the output from this attempt in Fiure 2:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/ValueError.png) **Figure 2. Output from running Listing 4 and inputting "h" for fltV1**

As you see in Figure 2, since I printed type(e), I now know that this incorrect entry caused a ValueError type of exception. So, now I can add a message for this specific case:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 5

# Description: Code fragment showing my new specific exception and message for the ValueError

# Produced when I ran listing 4 and entered incompatible data

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

except ValueError as e:

print("Input data must be numeric!")

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

I experimented with breaking my program in different ways and ended up adding an exception for trying to divide by 0 (ZeroDivisionError). I experimented with multiplying instead of dividing, and tried multiplying very large numbers together to see if I could create an infinite value error (something I thought I remembered from a different programming language), but Python didn't error over that, it just returned the value "inf."

I also wrote my output to a file, and determined that if I tried to append data to a file that didn't already exist, I got a FileNotFoundError, so I added an exception for that as well. You can also look at Python's list of built in exceptions to try to anticipate which ones might be relevant for your program: <https://docs.python.org/3/library/exceptions.html#bltin-exceptions> [directs to an outside url]

**Pickling**

Pickling is the term used for working with binary files in Python. Pickling = serialization and unpickling = un-serialization. I had wondered why they were called "pickles" and found out that the name is mostly referring to preservation and storage. Much like cucumbers are preserved for long term storage when turned into pickles, so will the data be easier preserved if you pickle it. I found a helpful explanation of pickling on this website: <https://www.geeksforgeeks.org/understanding-python-pickling-example/> [directs to an outside url] The author explained what makes pickling useful:

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.

I also really enjoyed the example code I found on this website, because it helped me do one thing I was stumped about after watching the module video - it allowed me to print *every row* from a pickled object, instead of just the first row, that you get from using just print(pickle.load(data))

It did require me to make a database, but it didn't seem much different than a dictionary -- it seems just like a set of dictionaries where the dictionaries have names? So conceptually it made sense to me and I was able to play around with it. I made a database called "Household" with all the human members of my household plus some identifying details, pickled that, and then printed all of it using a loop.

#--------------------------------------------------------------------------#

# Title: Listing 6

# Description: Pickling Example

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Modified a script taken from

# https://www.geeksforgeeks.org/understanding-python-pickling-example/

#--------------------------------------------------------------------------#

import pickle

def storeData():

# initializing data to be stored in db

Meredith = {'id': 1, 'name': 'Meredith Shapiro','age': 36, 'sex': 'F'}

Jeremy = {'id': 2 , 'name': 'Jeremy Shapiro', 'age': 42, 'sex': 'M'}

Edie = {'id': 3, 'name': 'Edie Shapiro', 'age': 8, 'sex': 'F'}

Calvin = {'id': 4, 'name': 'Calvin Shapiro', 'age': 5, 'sex': 'M'}

# database

household = {}

household['M'] = Meredith

household['J'] = Jeremy

household['E'] = Edie

household['C'] = Calvin

# Its important to use binary mode

dbfile = open('examplePickle', 'wb')

# source, destination

pickle.dump(household, dbfile)

dbfile.close()

def loadData():

# for reading also binary mode is important

dbfile = open('examplePickle', 'rb')

db = pickle.load(dbfile)

for keys in db:

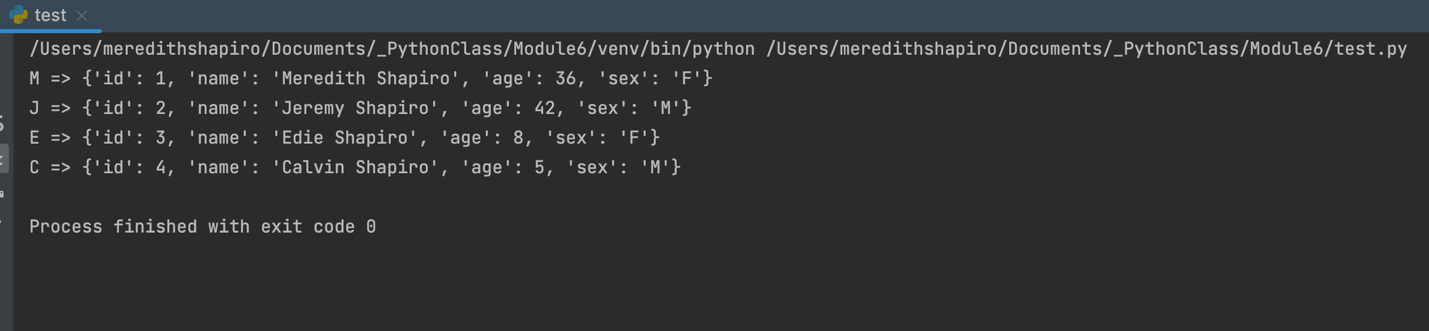
print(keys, '=>', db[keys])

dbfile.close()

storeData()

loadData()

The key things to remember are that if you are trying to pickle to a file, the file open modes are all suffixed with b (for binary), so "rb" instead of "r" for read, "ab" for append, and "wb" for write. And all examples I looked at used the key functions of pickle.dump() for turning info into a pickle and dumping into a file, and pickle.load() for reading from a pickle.

Here is the output from Listing 6: [](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/PickleOutput.png) **Figure 3. Output from running Listing 6**

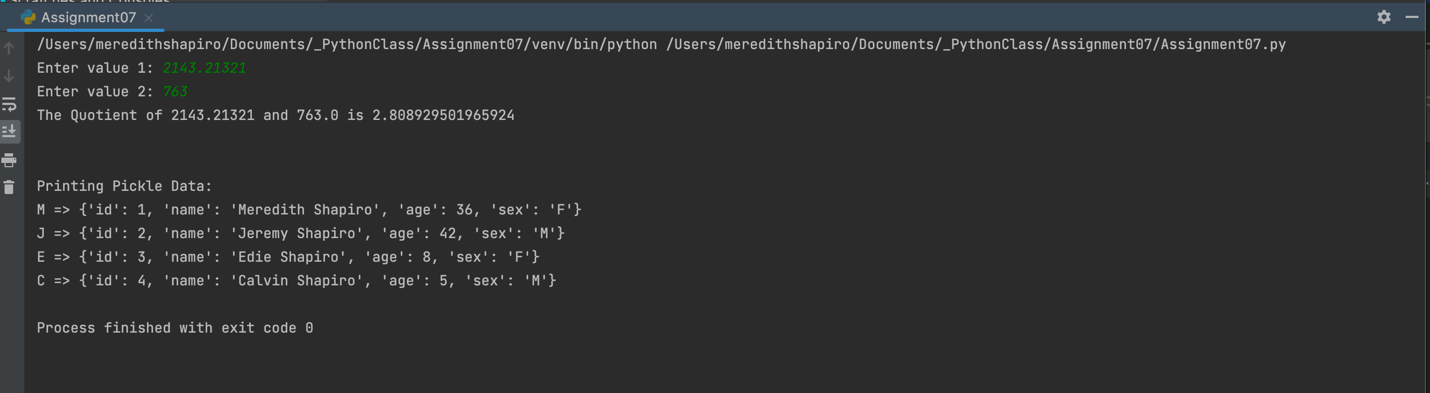
In Figure 3 you can see that I successfully printed all the dictionaries from the database that I pickled, which I appreciated for troubleshooting.

**Exceptions while pickling**

I added my pickling code in to my homework file with the error-handling code. I then tried to break the pickling process, which was pretty easy by forgetting to add the b to the file commands. I generated a few additional exceptions and added comments for them as well (TypeError, UnicodeDecodeError, and a special pickle error, pickle.UnpicklingError).

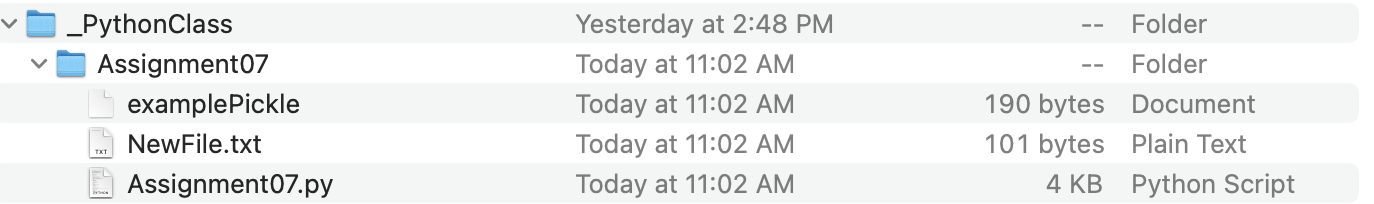
**Running the Script**

First, here is what it looks like when I run the script correctly in Pycharm:

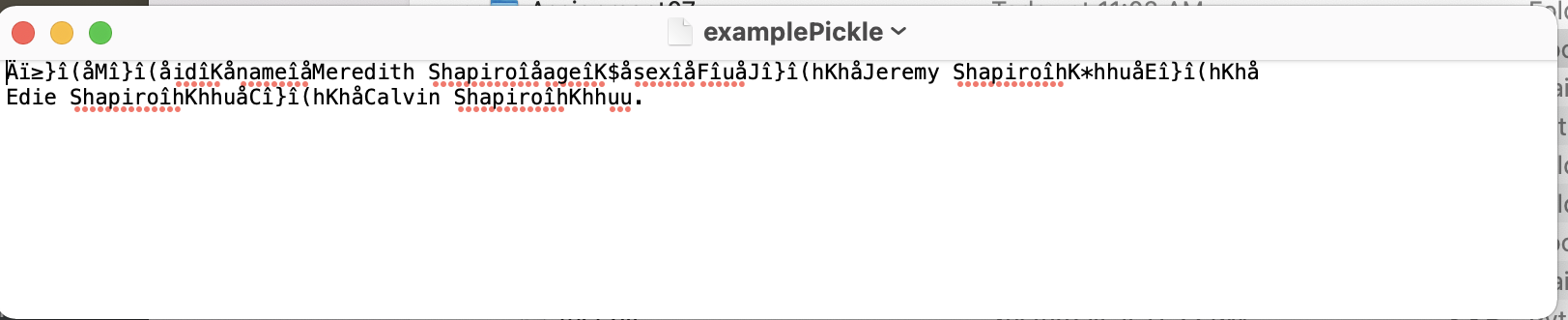
[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Successful%20Run%20of%20Script%20in%20Pycharm.png) **Figure 4. Output from running script in Pycharm**

As you see in Figure 4, I can run the script and it performs my function that takes two user input values and divides them, then prints a message about the quotient. Next it creates a binary file and reads from it. There are some steps I didn't print messages for (saving the numbers and quotient to a text file, the creation of the pickle) because I didn't feel this was essential.

I can see that I did create a text file with the twoVals output, and, more importantly, the binary file called examplePickle:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/File%20contents%20-%20docs%20created.png) **Figure 5. Contents of folder, showing that files were created while running script**

Let's open the binary file we created, as seen in Figure 5.

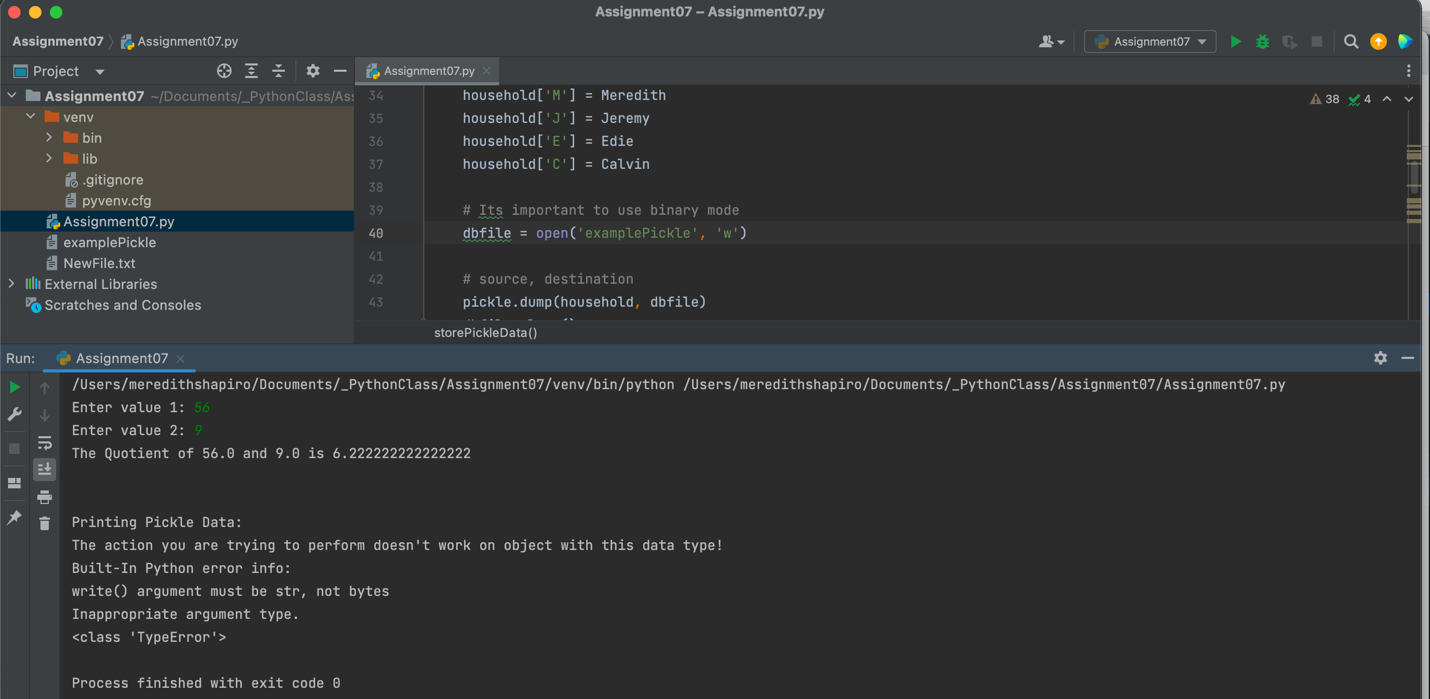
[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/PickleContents.png) **Figure 6. examplePickle file opened in TextEditor**

As you see in Figure 6, a binary file isn't really easy to read by a human. I'm so glad we can display the contents using pickle.load()!

Now, what happens if I forget the "b" in the statement

dbfile = open('examplePickle', 'wb')

I took off the b, as if I was trying to just write to a regular file. Then I tried to use pickle.dump() to the text file. Here is the error message I get:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Running%20in%20Pycharm%20with%20type%20error.png) **Figure 7. Error from trying to pickle.dump() to a text file**

In Figure 7, you see I created a custom message for this type of error (TypeError). This message may seem to generic - why not say something pickle specific, like "you need to use 'wb' to write to a binary file when pickling!"? I intially did have a more specific message like that, but in the course of testing my program and trying to create errors, I found that any kind of data type mismatch can create this error, so my specific message became *more* confusing! So, I tried to generalize it somewhat to make sense for multiple scenarios that could cause a TypeError.

Finally, I can run the script in Terminal. Here is output from me running it, first with an error and then correctly:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Running%20in%20Terminal%20wrong%20and%20right.png) **Figure 8. Output from running script in Terminal**

As you see in Figure 8, I initially tried to enter a string for my second value, which gave me one of my specific error messages.

**Summary**

In this document, you can see how we can work with Python's built in exception class to get info about what type of error has been created, and then use that info to craft meaningful messages for a user. I really enjoyed trying to intuit user behavior, and seeing what happened when I tried to break my program. We also learned how to "pickle" data into binary files for storage/preservation purposes.

**Footer**

© 2022 GitHub, Inc.

**Footer navigation**

* [Terms](https://docs.github.com/en/github/site-policy/github-terms-of-service)
* [Privacy](https://docs.github.com/en/github/site-policy/github-privacy-statement)
* [Security](https://github.com/security)
* [Status](https://www.githubstatus.com/)
* [Docs](https://docs.github.com/)
* [Contact GitHub](https://support.github.com/?tags=dotcom-footer)
* [Pricing](https://github.com/pricing)
* [API](https://docs.github.com/)
* [Training](https://services.github.com/)
* [Blog](https://github.blog/)
* [About](https://github.com/about)

You have no unread notifications

Top of Form

Pin

Bottom of Form

* Unwatch 1
* Fork 0

Top of Form

 Star 0

Bottom of Form

* [**Code**](https://github.com/meshapiro/IntroToProg-Python-Mod07)
* [Issues](https://github.com/meshapiro/IntroToProg-Python-Mod07/issues)
* [Pull requests](https://github.com/meshapiro/IntroToProg-Python-Mod07/pulls)
* [Actions](https://github.com/meshapiro/IntroToProg-Python-Mod07/actions)
* [Projects](https://github.com/meshapiro/IntroToProg-Python-Mod07/projects)
* [Wiki](https://github.com/meshapiro/IntroToProg-Python-Mod07/wiki)
* [Security](https://github.com/meshapiro/IntroToProg-Python-Mod07/security)
* [Insights](https://github.com/meshapiro/IntroToProg-Python-Mod07/pulse)
* [Settings](https://github.com/meshapiro/IntroToProg-Python-Mod07/settings)

 main

[**IntroToProg-Python-Mod07**](https://github.com/meshapiro/IntroToProg-Python-Mod07)/[docs](https://github.com/meshapiro/IntroToProg-Python-Mod07/tree/main/docs)/**index.md**

[Go to file](https://github.com/meshapiro/IntroToProg-Python-Mod07/find/main)

[@meshapiro](https://github.com/meshapiro)

[**meshapiro**](https://github.com/meshapiro) [Update index.md](https://github.com/meshapiro/IntroToProg-Python-Mod07/commit/3a3e88d9ea195e38e2a66a349418ac1beaebb0ec)

Latest commit [3a3e88d](https://github.com/meshapiro/IntroToProg-Python-Mod07/commit/3a3e88d9ea195e38e2a66a349418ac1beaebb0ec) 3 minutes ago[**History**](https://github.com/meshapiro/IntroToProg-Python-Mod07/commits/main/docs/index.md)

**1** contributor

224 lines (160 sloc)  13.3 KB

[Raw](https://github.com/meshapiro/IntroToProg-Python-Mod07/raw/main/docs/index.md)[Blame](https://github.com/meshapiro/IntroToProg-Python-Mod07/blame/main/docs/index.md)

Top of Form

Bottom of Form

Top of Form

Bottom of Form

**Error Handling and Pickling**

*MShapiro, 8/25/22*

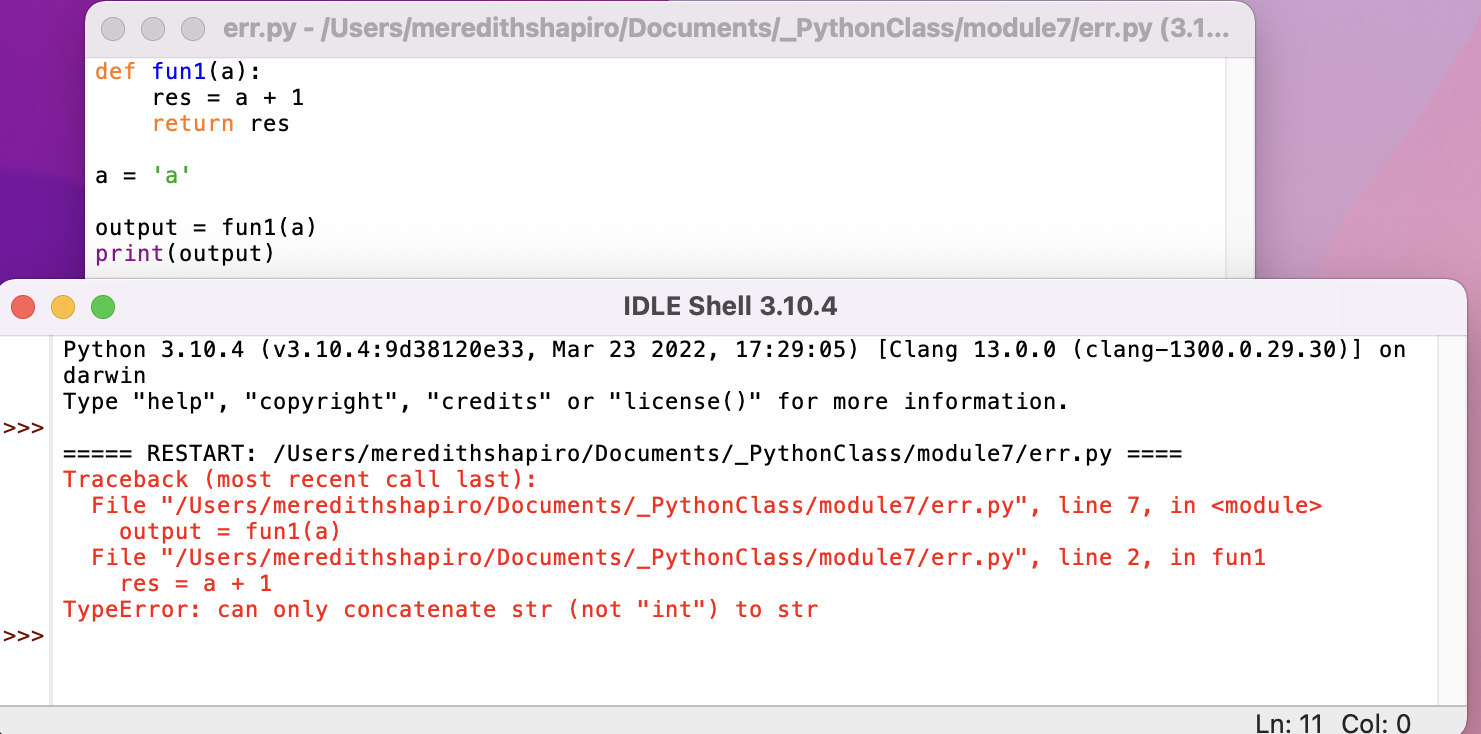
*Assignment07*

[*https://github.com/meshapiro/IntroToProg-Python-Mod07*](https://github.com/meshapiro/IntroToProg-Python-Mod07)

**Introduction**

In this markdown document, I will explain the basics of how to display custom error messages for users of our Python scripts. I will also cover pickling (binary files).

**Error Handling**

When user input is required as part of running a script, this opens the door for user error. Any user can, and most likely will, introduce errors while running a program. As we know from developing code, when you make an error in Python, you will see a system generated error message in red text. But, these messages are sometimes too technical or not informative enough for a user. [](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/error_message.png) **Figure 1. System-generated error message from running a function with incompatible data**

We can instead add a "try-except" block to our code, which will allow the program to capture errors, and display more informative error messages.

To use try-except, all the code you are running (the main body, not the section where you define functions) can be enclosed in "try: "

#--------------------------------------------------------------------------#

# Title: Listing 1

# Description: Excerpt of the "try" portion of my script

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#--------------------------------------------------------------------------#

try:

fltV1 = float(input("Enter value 1: "))

fltV2 = float(input("Enter value 2: "))

twoVals(fltV1, fltV2)

Make sure to indent everything that happens within the "try." After the main body of the script enclosed in "try: ", you will add "except: " to tell Python how to handle exceptions (errors). You could just print a generic error message, like this:

#--------------------------------------------------------------------------#

# Title: Listing 2

# Description: Example of a generic exception with message

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#--------------------------------------------------------------------------#

except Exception as e:

print("There was an error!")

That would solve the issue of the error message being too technical, but it doesn't help it be more informative. We can get the details of an error by using Python's Exception class:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 3

# Description: Code fragment showing how to get additional details on an exception, and its type

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

except Exception as e:

print(e, e.\_\_doc\_\_, type(e), sep='\n')

We can print the exception itself (e, the name we have given it) as well as more detailed documentation, and the type of error - type(e).

Once we have these details, we can create separate Exceptions for each type of error that seems likely, and display custom messages for them. If we know the type of error, we can replace "Exception" in the "except Exception as e:" statement, with whatever the actual exception type is. For instance, I wrote a function that takes input for two numeric values and divides them. I wanted to see what would happen if I entered something non-numeric:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 4

# Description: Code fragment showing how to get additional details on an exception, and its type

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

try:

fltV1 = float(input("Enter value 1: "))

fltV2 = float(input("Enter value 2: "))

twoVals(fltV1, fltV2)

fltV1, fltV2, fltQuot = twoVals(fltV1, fltV2) # unpack tuple

output = displayVals(fltV1, fltV2, fltQuot)

print(output)

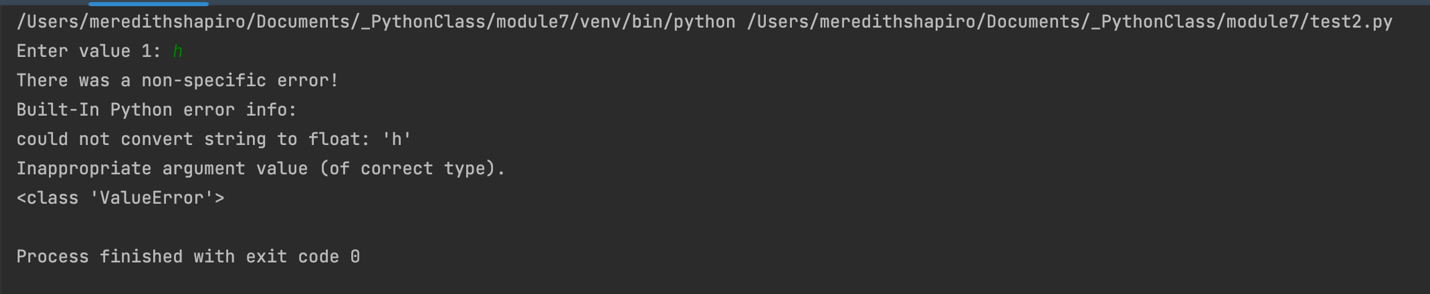
except Exception as e:

print("There was a non-specific error!")

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

I tried running this but, when asked for my input for the first value, I entered "h". You can see the output from this attempt in Fiure 2:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/ValueError.png) **Figure 2. Output from running Listing 4 and inputting "h" for fltV1**

As you see in Figure 2, since I printed type(e), I now know that this incorrect entry caused a ValueError type of exception. So, now I can add a message for this specific case:

#-------------------------------------------------------------------------------------------------#

# Title: Listing 5

# Description: Code fragment showing my new specific exception and message for the ValueError

# Produced when I ran listing 4 and entered incompatible data

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Wrote script

#-------------------------------------------------------------------------------------------------#

except ValueError as e:

print("Input data must be numeric!")

print("Built-In Python error info: ")

print(e, e.\_\_doc\_\_, type(e), sep='\n')

I experimented with breaking my program in different ways and ended up adding an exception for trying to divide by 0 (ZeroDivisionError). I experimented with multiplying instead of dividing, and tried multiplying very large numbers together to see if I could create an infinite value error (something I thought I remembered from a different programming language), but Python didn't error over that, it just returned the value "inf."

I also wrote my output to a file, and determined that if I tried to append data to a file that didn't already exist, I got a FileNotFoundError, so I added an exception for that as well. You can also look at Python's list of built in exceptions to try to anticipate which ones might be relevant for your program: <https://docs.python.org/3/library/exceptions.html#bltin-exceptions> [directs to an outside url]

**Pickling**

Pickling is the term used for working with binary files in Python. Pickling = serialization and unpickling = un-serialization. I had wondered why they were called "pickles" and found out that the name is mostly referring to preservation and storage. Much like cucumbers are preserved for long term storage when turned into pickles, so will the data be easier preserved if you pickle it. I found a helpful explanation of pickling on this website: <https://www.geeksforgeeks.org/understanding-python-pickling-example/> [directs to an outside url] The author explained what makes pickling useful:

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.

I also really enjoyed the example code I found on this website, because it helped me do one thing I was stumped about after watching the module video - it allowed me to print *every row* from a pickled object, instead of just the first row, that you get from using just print(pickle.load(data))

It did require me to make a database, but it didn't seem much different than a dictionary -- it seems just like a set of dictionaries where the dictionaries have names? So conceptually it made sense to me and I was able to play around with it. I made a database called "Household" with all the human members of my household plus some identifying details, pickled that, and then printed all of it using a loop.

#--------------------------------------------------------------------------#

# Title: Listing 6

# Description: Pickling Example

# ChangeLog: (Who, When, What)

# MShapiro, 8/25/22, Modified a script taken from

# https://www.geeksforgeeks.org/understanding-python-pickling-example/

#--------------------------------------------------------------------------#

import pickle

def storeData():

# initializing data to be stored in db

Meredith = {'id': 1, 'name': 'Meredith Shapiro','age': 36, 'sex': 'F'}

Jeremy = {'id': 2 , 'name': 'Jeremy Shapiro', 'age': 42, 'sex': 'M'}

Edie = {'id': 3, 'name': 'Edie Shapiro', 'age': 8, 'sex': 'F'}

Calvin = {'id': 4, 'name': 'Calvin Shapiro', 'age': 5, 'sex': 'M'}

# database

household = {}

household['M'] = Meredith

household['J'] = Jeremy

household['E'] = Edie

household['C'] = Calvin

# Its important to use binary mode

dbfile = open('examplePickle', 'wb')

# source, destination

pickle.dump(household, dbfile)

dbfile.close()

def loadData():

# for reading also binary mode is important

dbfile = open('examplePickle', 'rb')

db = pickle.load(dbfile)

for keys in db:

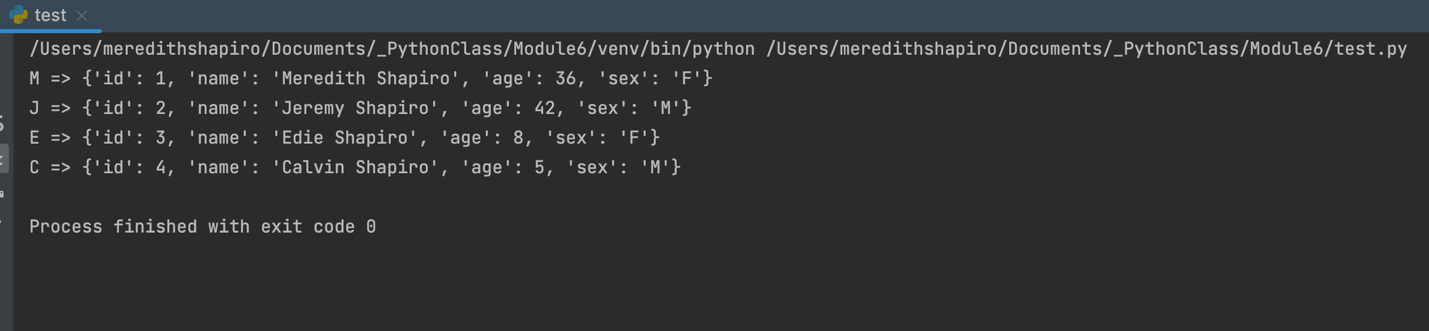
print(keys, '=>', db[keys])

dbfile.close()

storeData()

loadData()

The key things to remember are that if you are trying to pickle to a file, the file open modes are all suffixed with b (for binary), so "rb" instead of "r" for read, "ab" for append, and "wb" for write. And all examples I looked at used the key functions of pickle.dump() for turning info into a pickle and dumping into a file, and pickle.load() for reading from a pickle.

Here is the output from Listing 6: [](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/PickleOutput.png) **Figure 3. Output from running Listing 6**

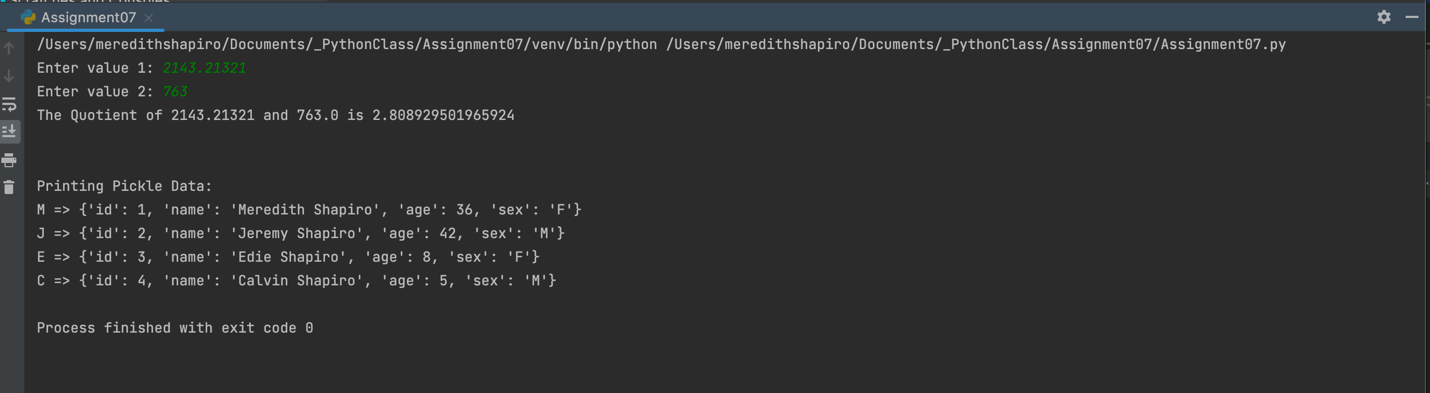
In Figure 3 you can see that I successfully printed all the dictionaries from the database that I pickled, which I appreciated for troubleshooting.

**Exceptions while pickling**

I added my pickling code in to my homework file with the error-handling code. I then tried to break the pickling process, which was pretty easy by forgetting to add the b to the file commands. I generated a few additional exceptions and added comments for them as well (TypeError, UnicodeDecodeError, and a special pickle error, pickle.UnpicklingError).

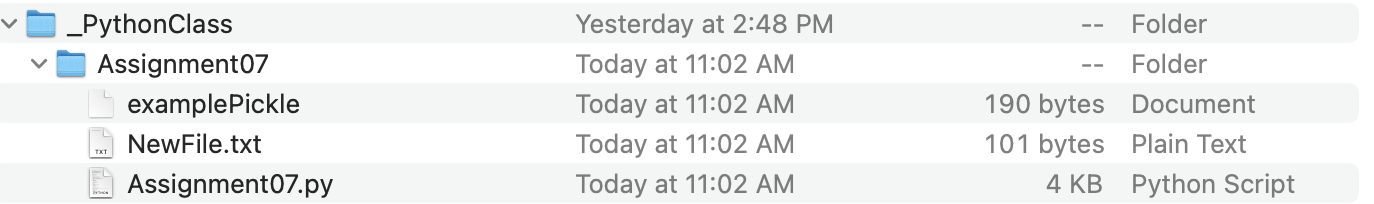
**Running the Script**

First, here is what it looks like when I run the script correctly in Pycharm:

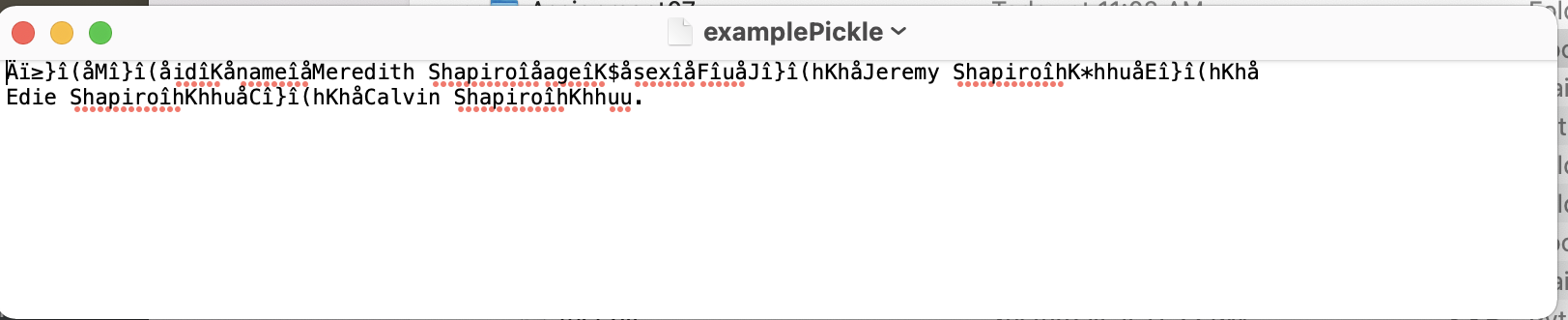
[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Successful%20Run%20of%20Script%20in%20Pycharm.png) **Figure 4. Output from running script in Pycharm**

As you see in Figure 4, I can run the script and it performs my function that takes two user input values and divides them, then prints a message about the quotient. Next it creates a binary file and reads from it. There are some steps I didn't print messages for (saving the numbers and quotient to a text file, the creation of the pickle) because I didn't feel this was essential.

I can see that I did create a text file with the twoVals output, and, more importantly, the binary file called examplePickle:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/File%20contents%20-%20docs%20created.png) **Figure 5. Contents of folder, showing that files were created while running script**

Let's open the binary file we created, as seen in Figure 5.

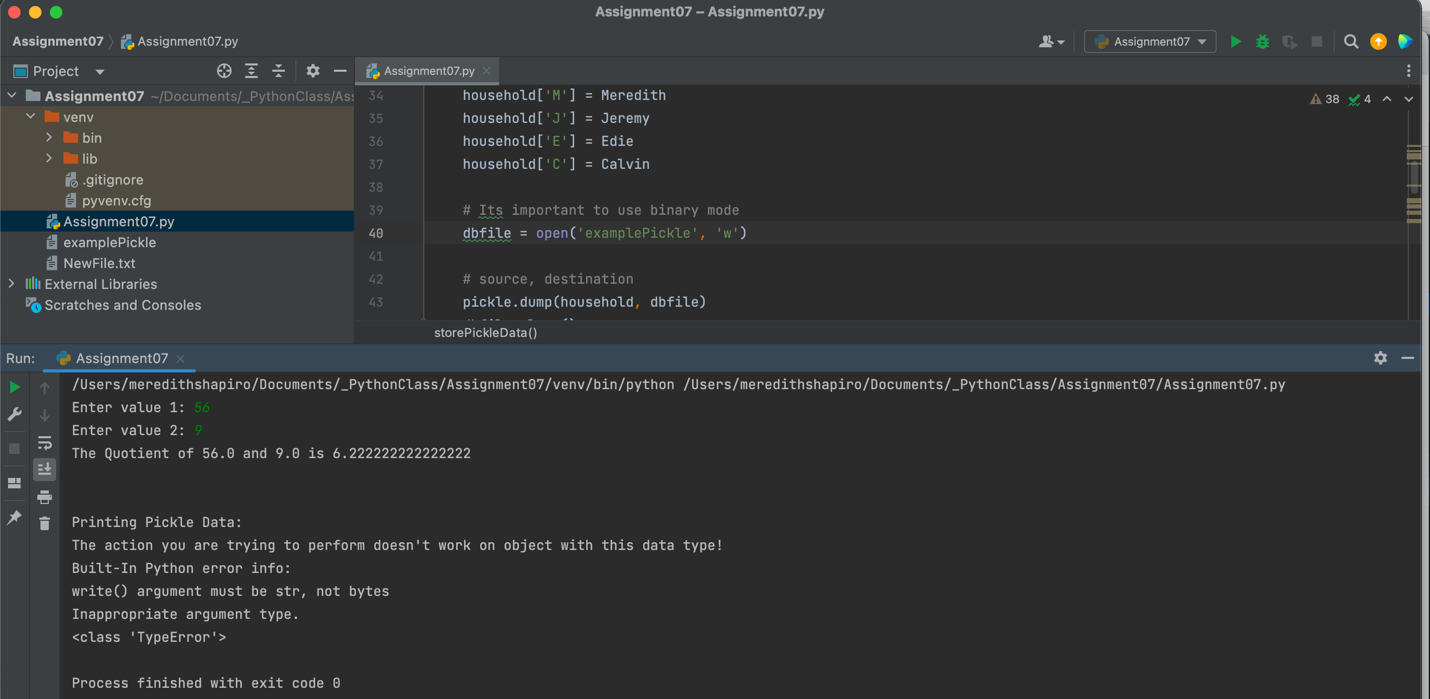
[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/PickleContents.png) **Figure 6. examplePickle file opened in TextEditor**

As you see in Figure 6, a binary file isn't really easy to read by a human. I'm so glad we can display the contents using pickle.load()!

Now, what happens if I forget the "b" in the statement

dbfile = open('examplePickle', 'wb')

I took off the b, as if I was trying to just write to a regular file. Then I tried to use pickle.dump() to the text file. Here is the error message I get:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Running%20in%20Pycharm%20with%20type%20error.png) **Figure 7. Error from trying to pickle.dump() to a text file**

In Figure 7, you see I created a custom message for this type of error (TypeError). This message may seem to generic - why not say something pickle specific, like "you need to use 'wb' to write to a binary file when pickling!"? I intially did have a more specific message like that, but in the course of testing my program and trying to create errors, I found that any kind of data type mismatch can create this error, so my specific message became *more* confusing! So, I tried to generalize it somewhat to make sense for multiple scenarios that could cause a TypeError.

Finally, I can run the script in Terminal. Here is output from me running it, first with an error and then correctly:

[](https://github.com/meshapiro/IntroToProg-Python-Mod07/blob/main/docs/Running%20in%20Terminal%20wrong%20and%20right.png) **Figure 8. Output from running script in Terminal**

As you see in Figure 8, I initially tried to enter a string for my second value, which gave me one of my specific error messages.

**Summary**

In this document, you can see how we can work with Python's built in exception class to get info about what type of error has been created, and then use that info to craft meaningful messages for a user. I really enjoyed trying to intuit user behavior, and seeing what happened when I tried to break my program. We also learned how to "pickle" data into binary files for storage/preservation purposes.

**Footer**

© 2022 GitHub, Inc.

**Footer navigation**

* [Terms](https://docs.github.com/en/github/site-policy/github-terms-of-service)
* [Privacy](https://docs.github.com/en/github/site-policy/github-privacy-statement)
* [Security](https://github.com/security)
* [Status](https://www.githubstatus.com/)
* [Docs](https://docs.github.com/)
* [Contact GitHub](https://support.github.com/?tags=dotcom-footer)
* [Pricing](https://github.com/pricing)
* [API](https://docs.github.com/)
* [Training](https://services.github.com/)
* [Blog](https://github.blog/)
* [About](https://github.com/about)

You have no unread notifications