Kasia Rozanska

Dec. 14, 2020

Foundations of Programming: Python

Assignment07

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

Error Handling and Pickling

# Introduction

In this demonstration we will ask for a user’s name and age, calculate it into seconds, and save it as a binary file. This will help us discuss error handling with try and except blocks, as well as pickling. You may follow the GitHub repository to see the full script and follow below to understand specific snippets.

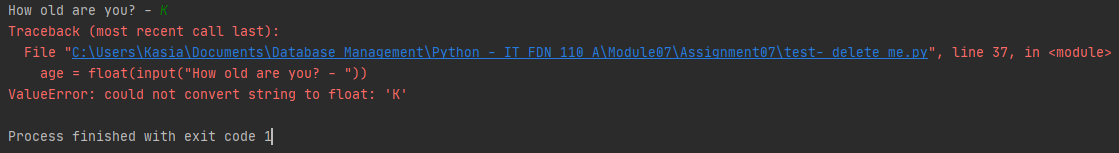
## Error Handling

As we have reviewed in previous demonstrations, you want to stay one step ahead of your user when asking for inputs. One way you do this is by controlling the data they can enter, for example, only allowing numeric characters in an age field. Another way is by controlling the error messages they receive, rather than the messages that Python would show. This allows for cleaner data and a user-friendly environment.

Look at the below code.

age = float(input("How old are you? - "))

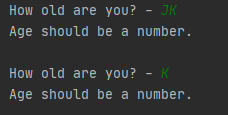
If the user does not enter a number, as our code expects, they will receive the below error. We understand it because we understand Python syntax, but this does not mean that your user will understand what they did incorrectly.



The try and except blocks, examples of structured error handling, were created to offer more consistency within programs and different developers. <https://www.youtube.com/watch?v=nlCKrKGHSSk> gives a quick overview of the four clauses in Python Exception handling. Below we will use the same code as above, but use try and except to help the user receive an error message that will guide them in the right direction. In this example we are replacing the specific exception ValueError message with “Age should be a number.”

while True:  
 try:  
 age = float(input("How old are you? - "))  
 except ValueError:  
 print("Age should be a number.\n")

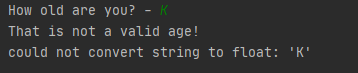
The code will first execute the *try clause*, and if no exception occurs then it will skip the *except clause*. See below how much cleaner the error is now?



<https://docs.python.org/3/library/exceptions.html> gives a great overview of all the exceptions that are built into Python. When an exception occurs, it may have an *argument*, which is an associated value. You can display this argument by specifying a variable after the exception.

try:  
 age = float(input("How old are you? - "))  
except Exception as e:  
 print("That is not a valid age!")  
 print(e)

Now the user will see exactly what created the error message. In the example below we have a simple version and we see that the issue is the “K” that was entered, but this error handling can become more useful with more advanced code.



You can also group multiple exception types by listing them in a single *except clause*, by creating a comma-separated group enclosed in a set of parentheses:

while True:  
 try:  
 age = float(input("How old are you? - "))  
 except (ValueError, TypeError):  
 print("Age should be a number.\n")

## Creating Your Own Exceptions

In the below code we have examples of how we can create our own exceptions and exception classes. This allows for more flexibility if we know we have parameters we need to meet, like when asking a user to create a password. For this demo I have created the rules “name should be longer than one character,” and “name should only be alphabetic characters.” A more detailed explanation can be found at <https://www.programiz.com/python-programming/user-defined-exception>.

class Error(Exception):  
 *""" Base class for other exceptions"""* pass  
  
  
class ValueTooSmallError(Error):  
 *""" Raised when the input value is too small"""* pass  
  
  
class ValueIncludesNumbersError(Error):  
 *""" Raised when the input includes a number"""* pass  
  
while True:  
 try:  
 name = str(input("What is your name? - ")).strip()  
 if len(name) < 2:  
 raise ValueTooSmallError  
 elif name.isalpha() is False:  
 raise ValueIncludesNumbersError  
 except ValueTooSmallError:  
 print("Your name must be longer than one character.\n")  
 except ValueIncludesNumbersError:  
 print("You're not \"X Æ A-12\"! Enter a name that only has alphabetic characters.\n")  
 except (TypeError, ValueError):  
 print("Something went wrong! Retype your name.\n")  
 else:  
 break

## Pickling

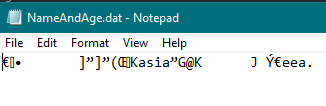
Up until now we have written to and read from text files. These are convenient because they can be manipulated in any text editor, but they do not allow for storing more complex information, like image or audio data. Pickling allows for us to preserve data so that it can be used by binary files. Binary files store data in the form of a sequence of bytes and allow developers to have the power to design custom file formats. To use pickling in your code you first import the **pickle** module.

import pickle

Like text files, binary files follow a similar syntax to write to and read from, generally you just add a “b” to the file access mode (see next two screenshots of code). To store the data we use the **pickle.dump** method. <https://pythontic.com/modules/pickle/dumps> provides a great explanation of the method signature and overview.

file = open(file\_name, "wb")  
pickle.dump(list\_of\_rows, file)  
file.close()

For this demo I have used file\_name = “NameAndAge.dat.” Now that our data was written to this document you will see it displayed in the following way:



To read the data we use the **pickle.load** method.

print("\nUnpickling list.")  
file = open(file\_name, "rb")  
list\_of\_rows = pickle.load(file)  
print(list\_of\_rows)

In order to see this data I have included both print statements. Below is how it displays to the user.



# Summary

Follow the full script from the GitHub repository and see for yourself how exceptions and pickling work. Create your own rules for a user’s input and write code that will force the user to enter data correctly. Practicing this logic will help you in the future when you are designing a new program for users to navigate. Finally, write and read from a binary file and confirm that it is as simple as working with a text file. Once you know how to do this, you’ll be ready to learn more about all the opportunities with pickled data.