Kyle Banfill

3/7/2022

Foundations of Programming: Python

Assignment 07

<github>

Pickling and Structured Error Handling

Introduction

In this module, we learned more about working text and binary files, specifically reading and appending data to them. We also explored structured error handling

Questions

1. What are the benefits of putting built-in Python command into functions?

Multiple actions can be performed with one function call.

1. What are the benefits of using structured error handling?

Structured error handling allows you to create errors that are much more readable to users when users may introduce new bugs into your code as a result of their inputs.

1. What are the differences between a text file and a binary file?

Text files save data as text strings that are human readable while binary saves them in smaller non-human readable formats.

1. How is the Exception class used?

Exception classes are used to automatically fill information related to the error that caused the exception.

1. How do you "derive" a new class from the Exception class?

Along the structured hierarchy of Exception classes, there are more specific “derived” classes that inherit the properties of the overarching parent, or base, class.

1. When might you create a class derived from the Exception class?

If you wanted to dissuade the user from using specific values that you find undesirable, but are otherwise sound in the code, you might want to create a class with a custom error message.

1. What is the Markdown language?

Markdown is a formatting language.

1. How do you use Markdown on a GitHub webpage?

You use Markdown to format your text and then GitHub uses Jekyll to automagically convert it into a static HTML page.

Pickling and Structured Error Handling Code

For this week’s module, we created script to illustrate pickling and structured error handling.

Text

Description automatically generated

Text

Description automatically generated

I’ve mashed the two of them together into one script.

Text

Description automatically generated

In the figure above, we import code from another code file first. Then, we create our try-except structure. We’ve raised custom errors where entering numbers for pickle names or ranking dill pickles number 1 will return errors.

Text

Description automatically generated

Text

Description automatically generated

I originally had this just as try-except block and then continued afterward, but I was unable to get the code to stop if it hit any of the above custom errors. As a result, I decided to use an else block. This way, the code would only continue if it didn’t run into the errors. I’m unsure why hitting the except block didn’t stop the code entirely, but this is a satisfactory workaround for me. Continuing with runtime errors, inputs we don’t want the user to use, didn’t seem right.

As an aside, I do not actually think dill pickles are terrible. It’s just a joke.

Text

Description automatically generated

We create another try-except block here to capture any errors that might occur when trying to pickle and write the data. I’m unsure what kind of specific errors might occur, so I’ve left it as a non-specific error.

Text

Description automatically generated

And then we used another try-except block here for the unpickling. I thought that an error may occur if the file were erased after the previous step, so I included an except block for FileNotFoundError as well as a non-specific error block.

Text

Description automatically generated

If no errors are detected, we move onto the actual pickling section where we take the list we created and write it a file in binary. By pickling, we serialize the data so that it can be sent to anyone. We won’t be sending this anywhere in this exercise, but it’s handy to know. Next, we unpickle it to translate it back into readable text that is printed.

Conclusion

In this module, we learned to structure code around capturing errors in specific blocks with try-except. We also learned how to work with binary files and convert information between binary and text formats. Lastly, we learned about markdown to format GitHub pages to help with sharing our data.