Error handling

Estimated time for completion: 45 minutes

Overview:

In this lab exercise, you will explore error handling in Python. We will take a somewhat contrived example of a code file line counter module, which has a set of possible errors. This module will read a directory full of Python code and tell you the number of lines of non-commented code. The contrived part has to do with the errors it throws, but it will be instructive to work with them nonetheless. Note that while the file APIs are in heavy use in this project, you do not need to understand them (yet) to be successful with this lab. We will cover file APIs elsewhere.

These instructions assume you have **Python 3** installed. You can use any OS you choose (OS X, Windows, or Linux).

Goals:

- Catch errors and report on them
- Distinguish between error types
- Define and raise custom exceptions

Part 1 – Experiment with the Python line counter

Open the **program.py** and **loc.py** files (or project if you are using PyCharm) in **Error_Handling/Before/error_handling_lab_before**. Notice that the program.py file depends upon the loc.py file. Run program.py and observe that it crashes. **Without** debugging this or opening loc.py to inspect it, use the traceback to answer the following questions:

1. Which line in **program.py** caused the error?

2. Which method in PythonLineCounter actual threw the error?
3. Why did this crash?

Part 2 – Catching some errors

The first thing we want to do is stop our program.py from actually crashing outright. We want to catch the errors and report them to the user. To do this, you will modify the **count_lines** method in **program.py**.

Add a generic error handling section within that method and stop it from crashing. Print out as much detail as you can for the error (should be little at first).

Because all the remaining errors in PythonLineCounter just raise Exception, we cannot do much with this. Now you should open up loc.py and fix this. You should define the following exceptions:

Exception	**Associated data**
InvalidFileTypeError	File name, extension
FileTooLargeError	File name, number of lines

Recall to create exceptions, they

- are simply standard classes
- derive (directly or indirectly) from Exception an have standard constructors (__init__ methods)
- they should end in Error

Update the PythonLineCounter to use these exceptions in the correct places.

Your final output should look something like this (modulo the variations in folder structure). It is very important that your process exits with code 0. That means you did not let the exceptions escape.

```
Counting python files and lines in: ~/this_is_a_missing_folder
```

```
ERROR (file not found): ~/this_is_a_missing_folder



Counting python files and lines in: labs/06_Error_Handling/input_files/game

Number of files in folder: 5

Number of lines in all files: 165



Counting python files and lines in: labs/06_Error_Handling/input_files/bad

Number of files in folder: 1

ERROR (invalid source file type): The file extension .cs of
labs/06_Error_Handling/input_files/bad/board.cs is not allowed.



Counting python files and lines in: labs/06_Error_Handling/input_files/bad2

Number of files in folder: 1

ERROR (file too large): Too many lines (10,323) in file
labs/06_Error_Handling/input_files/bad2/program.py.



Process finished with exit code 0
```