Exceptions with Files

Sometimes a program crashes unexpectedly, with an error message that looks similar to this:

```
Enter filename: data.txt
Traceback (most recent call last):
  File "/home/jgarvin/code/file_reader.py", line 2, in <module>
    f = open(file_name, "r")
FileNotFoundError: [Errno 2] No such file or directory: 'data.txt'
```

Program execution stops due to an **exception**, in this case a `FileNotFoundError`. In programming terms, we say that an exception was "thrown." In order to prevent the program from abruptly halting, it is possible to "catch" the exception and write code that will remedy the problem. Exceptions can also be forced using the `raise` keyword, but that is seldom necessary in this course.

To test whether some code will throw an exception, use the `try` keyword. Code inside of a `try` block may throw an exception. If it does, the first `except` clause that matches the exception type has its code executed. If no exceptions are thrown, code in an `else` clause (optional) may execute. A `finally` clause (optional) will always execute, whether an exception was thrown or not.

It is not mandatory to include an exception type in an `except` clause, but doing so will catch *all* exceptions (keyboard interrupts, inappropriate values, type-related issues, and so forth), which may not be appropriate. It is good practice to be specific.

There are many types of exceptions that can be caught in Python. For files, it is usually a good idea to catch `FileNotFoundError` exceptions, which will be thrown if a file opened for reading is not found. Other exceptions, such as input/output issues, are covered by the `OSError` exception. `FileNotFoundError` itself is a special case of `OSError`, but it may be preferable to have separate code for these things.

Typical Structure For Handling File-Related Exceptions

```
try:
      f = open(FILENAME, MODE)
      try:
            # FILE-PROCESSING GOES HERE
            # OTHER EXCEPTIONS MAY OCCUR HERE TOO!
      except EXCEPTION TYPE:
            # HANDLE OTHER EXCEPTIONS HERE
      finally:
            f.close()
except FileNotFoundError:
      print("File not found.")
except OSError:
      print("There was a problem with the file.")
```

Exceptions with Files

Answer the following questions.

1. What are some reasons why a `FileNotFoundError` exception might occur?

2. Although some resources suggest using the `finally` block to "clean up" by closing files and handling other related tasks, why might line 9 in the code below cause a crash?

   ```
   try:
         f = open("myfile.txt", "r")
   except FileNotFoundError:
         print("No such file exists.")
   else:
         for line in f:
            print(line)
   finally:
         f.close()
   ```

3. Why does the following code crash? How could you fix it?

   ```
   try:
         f = open("myfile.txt", "r")
   except:
         print("An unspecified error occurred.")
   except FileNotFoundError:
         print("No such file exists.")
   else:
         print("File successfully opened.")
         f.close()
   ```

Write programs that accomplish each task. Use proper conventions for variable names, input prompts, output statements, and program structure. You may assume that all files contain data in the formats specified. You should download the file `file_exceptions_files.zip` for testing.

4. Ask the user to specify a filename, then read and display the contents of the file. Your program should not crash if the file does not exist, or if it is unreadable.

5. The file `files.txt` contains a listing of filenames. Each of these files may or may not exist. Read the contents of `files.txt`, and generate counts of the number of files that exist, and those that do not. If a file exists, display its contents. As always, your program should not crash.

6. Python will often throw a `UnicodeDecodeError` exception when trying to read a binary file as a text file. Write a program that asks the user to provide the name of a text file, then read and display its contents. If a `UnicodeDecodeError` is thrown, output a message indicating that the file type is invalid.