HW08

Part 1: Date Arithmetic

Date arithmetic is far more difficult than you might imagine. For example, what is the date three days after Feb 27, 2000? It's not Feb 30. Was 2000 a leap year? Given two arbitrary dates, how many days are between those two dates?

Write a program to use Python's datetime module to answer the following questions:

1.1 What is the date three days after Feb 27, 2000?

1.2 What is the date three days after Feb 27, 2017?

1.3 How many days passed between Jan 1, 2017 and Oct 31, 2017?

You may write a simple program that just calls the appropriate datetime methods to answer the questions. You need not include automated test for part 1.

Part 2: Field separated file reader

Reading text files with a fixed number of fields, separated by a pre-defined character, is a common task. For example, .CSV files (comma separated values) are common in Windows applications while Unix data files are typically separated by the '|' (vertical bar) character.

Write a generator function to read text files and return all of the values on a single line on each call to next(). Your generator must meet the following requirements:

1. You must implement a **generator** that reads the file one line at a time and yields the values for that line. You should not read the entire file into memory.
2. The generator definition should include parameters for:
   - the path of the file to be read
   - the number of fields expected on each line
   - an optional parameter to specify the field separator which defaults to comma (',') (that wasn't intended as an emoji...)
   - an optional parameter that defaults to False to specify if the first line in the file is a header line
3. The generator should raise a FileNotFound exception if the specified file can't be opened for reading
4. The generator should raise a ValueError exception if a line in the file doesn't have exactly the expected number of fields. The exception message should **include the line number** in the file where the problem occurred, e.g. "ValueError: 'foo.txt' has 3 fields on line 26 but expected 4"
5. The ***n*** fields from each line should be yielded as a tuple

6. If the call to the generator specified a header row, then the first call for data should return the second row from the file, skipping over the header row

Here's an example of using the generator to read a vertical bar separated file with a header row and 3 fields:

```
for cwid, name, major in file_reader(path, 3, sep='|', header=True):
    print("name: {} cwid: {} major: {}".format(name, cwid, major))
```

Be sure to include automated tests with at least one test file

Hint: the code is much shorter than the description of the requirements

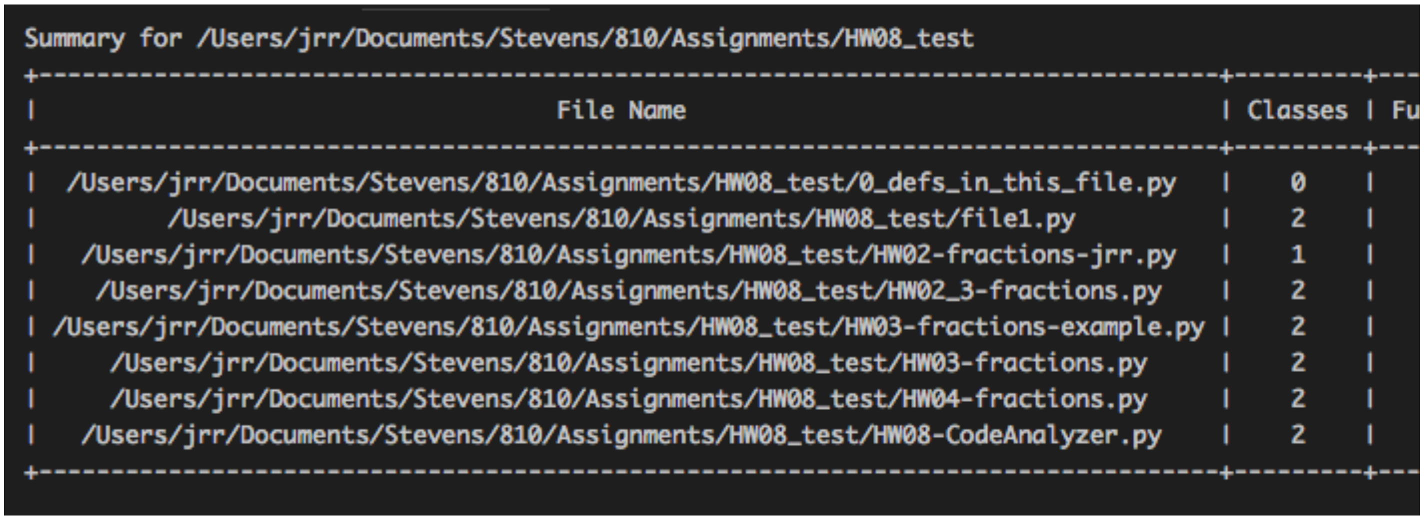
Part 3: Scanning directories and files

Python is a great tool for automating a number of tasks, including scanning and summarizing the files in a file system.

Write a Python program that given a directory name, searches that directory for Python files (i.e. files ending with .py). For each .py file, open each file and calculate a summary of the file including:

- the file name
- the total number of lines in the file
- the total number of characters in the file
- the number of Python functions (lines that begin with 'def ') - you ***should*** include class methods in the number of functions
- the number of Python classes (lines that begin with 'class ')

Generate a summary report with the directory name, followed by a tabular output that looks similar to the following:



```
Summary for /Users/jrr/Documents/Stevens/810/Assignments/HW08_test
+------------------------------------------------------------------------------+---------+---
|                                  File Name                                   | Classes | Fu
+------------------------------------------------------------------------------+---------+---
|  /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/0_defs_in_this_file.py  |    0    |
|         /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/file1.py         |    2    |
|   /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW02-fractions-jrr.py  |    1    |
|    /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW02_3-fractions.py   |    2    |
| /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW03-fractions-example.py |    2    |
|     /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW03-fractions.py     |    2    |
|     /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW04-fractions.py     |    2    |
|   /Users/jrr/Documents/Stevens/810/Assignments/HW08_test/HW08-CodeAnalyzer.py   |    2    |
+------------------------------------------------------------------------------+---------+---
```

Keep in mind that you may see a file, but you may not be able to read it.
In that case, just print the file name and note that it could not be opened.

Your program should use exceptions to protect the program against unexpected events.

The PrettyTable module (Links to an external site.)Links to an external site. provides an easy way to create tables for output. You should use PrettyTable to format your table. See the lecture notes for an example.

You must also structure your program into two separate files:

1. the source code for your directory and file analyzer
2. a separate file that holds the Unittest test cases for your program.

Your test file should import the relevant objects from your source file and call unittest.main() to perform the tests.

You'll need to think about how to test this program with unittest. You do not need to automatically verify the output from the table but you should validate the various numeric values, e.g. number of classes, functions, lines, and characters.

Hints:

- Python's os module has several helpful functions, e.g. os.listdir(dir) lists all of the files in the specified directory. Note that the file names returned by listdir() do NOT include the directory name.
- os.chdir(directory) will change the current directory to the specified directory when your program runs.
- Be sure to handle the cases where the user enters an invalid directory name
- Test your code against the sample files in Canvas

Be sure to use unittest to test your program and follow the PEP-8 coding standards.