Week 2 Assignment - Python Standard Library

For this week's assignment, we will be going over the 5 modules that we reviewed in the Python Standard Library. This homework should be a bit more difficult than last week, but not too difficult. Starting next week, the lesson content and assignments will pick up the pace.

Overview

For this assignment, you will be reading data from a file located on the internet. The file is a listing of people's name, their birthdays and an ID number in CSV format. This file, however, has some problems with it. Mostly, some of the birthdays are in an invalid format. The other columns in the CSV will be guaranteed to be in a proper format, so your code can assume so.

Using the modules that we learned about, you will design a program to do two things: write an output file recording which lines in the file *cannot* be processed correctly due to an improperly formatted date; and allow a user to enter in an ID number and print out that person's information.

Useful Reminders

1. Read the assignment over a few times. At least twice. It always helps to have a clear picture of the overall assignment when understanding how to build a solution.
2. Think about the problem for a while, and even try writing or drawing a solution using pencil and paper or a whiteboard
3. Before submitting the assignment, review the "Functional Requirements" section and make sure you hit all the points. This will not guarantee a perfect score, however.

Part 0 - Prerequisites

1. Our first step will be to create a github repo for this assignment. Please call it **IS211_Assignment2**.
2. After that is completed, use git to clone this repository to a local folder in your development environment. All development will be in this directory.

Part I - Think about Design

When approaching programming problems, you should first step back and think about the problem at hand. What *exactly* are we trying to accomplish, and what would be a good structure for your program? How would you break the larger problem into a set of smaller steps? One thing to think about in this case, is how you will want to store this data after reading it from the CSV.

Part II - Download the Data

Write a function called ***downloadData***, which takes in a string called *url*. The purpose of this function is to download the contents located at the *url* and return it to the caller. You should use urllib2 for this. Do not catch any exceptions here, as this will be done later.

Part III - Process Data

Write a function called ***processData***, which takes the contents of the file as the first parameter, processes the file line by line, and returns a dictionary that maps a person's ID to a tuple of the form (name, birthday). The birthday needs to be a Datetime object, not a string. You will have to process the birthday, which has a format of dd/mm/yyyy, and convert it into a Datetime object (**NOTE:** the time portion of the Datetime object will be 0, since we are not setting a time, just a date; you *can* use the Date object instead, if you feel comfortable doing so).

As stated above, the birthday might not be in a correct format. This might be due to a typo in the data (i.e., a missing forward slash as in 0304/2014) or an invalid date (i.e. 04/13/2014). If this is the case, we want to log this error using the logging module. This function should get a logger using the name 'assignment2'. This logger will be configured later in the program. The log message should be sent to the ERROR level, and of the format: "Error processing line #<linenum> for ID #<id>", where <*line*> and <*id*> are the line number and ID, respectively, of the person in the CSV file that has a malformed date.

Food For Thought: *There are many ways to represent that data in the CSV file using the basic Python data types. Why did we choose to store the data in the form of a dictionary that maps IDs to tuples? No need to submit this, but think about why this was chosen. Feel free to post about this in the weekly discussion board.*

Part IV - Display / User Input

Write a function called ***displayPerson***, which takes in an integer called *id* as its first parameter, and a dictionary as its second parameter, called *personData*. The purpose of this function is to print the name and birthday of a given user identified by the input *id*. If there is not entry with the given *id*, then print "No user found with that id" instead. Otherwise the format should be "Person #<id> is <name> with a birthday of <date>", where <id> is the <*id*> the user requested, <*name*> is the name of the person (from the file) and <*date*> is the birthday of the user (formatted as *YYYY-MM-DD*; **NOTE:** there is no time being displayed).

Part V - Putting it all Together

We now need to piece all these functions together. Lets write a ***main()*** function that is called when our script runs.

1. First, your program should use the argparse module to allow the user to send a *--url* parameter to the script. This parameter is a string and is required by the program. If no *--url* is given, the program should exit.
2. Then we need to call the ***downloadData()*** function and save the results to a variable. Lets call this variable csvData. The URL to pass to ***downloadData()*** is the URL given as a parameter to the script via argparse. Also, we need to also ensure that if any exception is thrown by this function, that we catch it, print out an appropriate error message, and exit the program.
3. We need to set up a logger with the name 'assignment2', which should be configured to write to a single file called "errors.log". This can be done in a separate function, or done in the main program.

4. Next, we need to take csvData and pass it to the ***processData()*** function. The result of this call should be saved into a variable called personData. Remember, this is a dictionary that maps IDs to tuples.
5. Next, ask the user for an ID to lookup.
    a. If the user enters in a negative number or 0, exit the program
    b. If the user enters in a positive number, use ***displayPerson()*** to attempt printing out the user
    c. Keep asking the user for input until they enter in a negative number or 0 to exit

Functional Requirements:

In this section, we will outline the various behaviours the program ***must*** do in order to get credit, and how we will be checking that these behaviours work correctly:

1. The program must accept and use the --url parameter. We will use this to point to not only the file in this assignment, but to another file you will not be able to see beforehand. The program should give correct output in both cases
2. The program must write to a file named error.log.
3. Each line of this error file must be of the correct format as specified in Part III
4. The program must print out users in the correct format, as directed by the user input
5. The program must exit *only* when the user enters in a number <= 0