Balchunas

AP CSP: Problem Set 4

Specification

Submit a .py file named `your_name_pset4.py`. Your file should **NOT** execute any functions. You are just filling them in. When I run your program there should be no errors and no output. You may choose to test your functions as you go, but you may delete the function calls after you test or comment them out. Comment your problems with the problem number above the function header. Remember to import the random module to use the library's functions.

Classic Rock Data

Load in the classic rock data set to answer the following questions. I suggest creating a list of dictionary objects where `tracks = [{'title': 'Caught Up in You', 'artist': '.38 Special', 'year': '1982', 'playcount': '82'}, {'title': 'Fantasy Girl', 'artist': '.38 Special', 'year': '1981', 'playcount': '3'}, {'title': 'Hold On Loosely', 'artist': '.38 Special', 'year': '1981', 'playcount': '85'}]` and so on.

Problem 1

Write a function that returns a dictionary where the key is the year a song was released and the value is the number of plays for songs released in that year. This corresponds to the histogram shown in the article. Ensure that your values match up with this plot.

Problem 1b

Write a function that prints out a counter dictionary as a sideways histogram. Use stars or hashes for the height of the bars. You may choose to scale the plot, just print out the scale as well. This is so your plot does not wrap onto multiple new lines. An example is below.

```
73:#####
74:#######
75:##########
76:###########
77:####
78:########
------------
# = 50 counts
```

Problem 2

Write a function that returns a reverse sorted list of tracks where the first element is the most played track and the last element is the least played track. You may use Python's built in `sorted()` function to sort the list of dictionary objects!

Problem 2b

Recreate the first table in the article by printing the first twenty songs from the sorted list of songs in the format: Title, Artist, Year. Use a formatted string to print out the information.

Problem 3

Write a function that returns a reverse sorted list of tuples where the first element of the tuple is the artist name and the second is the number of plays that artist has.

Problem 3b

Recreate the second table in the article by printing the first twenty-five artists from the sorted list of artists in the format: Artist, Pct of All Plays. Note we don't have the city data, so we cannot recreate the last two columns.

Blog App

In this assignment you will use what you know about loading and saving files to create a small blogging app similar to Bluesky/Twitter. User information is stored in a comma separated value file `users.csv`, individual blog posts will be stored in another comma separated value file `posts.csv`, and information about a user following other users will be stored in another comma separated value file `follows.csv`.

I suggest the user information be stored as username, password, that blog information be stored as username, timestamp, title, content, and that follows information be stored as username, following. So a user following three other users would have three lines in the `follows.csv` file. Note that Python's `datetime` module provides a nice way to get the current time as a string. It's a bit cryptic, but makes sense once you look at it long enough. Try running the following code and see what it does. I suggest you use this module to store the timestamps of your blog posts in a nice format.

```
import datetime

curr_time = datetime.datetime.now()

"""
 This object can be formatted using the strftime method.
 strftime takes an argument of a string that you want to return
 but you can use %Y to insert the year, %m to insert the month, %d to insert the
day
  %H to insert the hour, and %M to insert the minute.
  These special characters are case-sensitive. If you want the text month instead
of the numerical month you can use %b.
"""
formatted_time = curr_time.strftime("%Y/%m/%d at %H:%M")
print(formatted_time)
```

As for loading the data into Python, use a dictionary reader from the built-in csv module. I suggest usernames and passwords be loaded as a dictionary with keys of `username` and values equal to the password. I suggest the post information be stored as a dictionary where the keys are specific usernames and the values are lists

of post objects. A post object is a dictionary with timestamp, title, and content. I suggest loading the follows data as a dictionary with keys equal to specific usernames and values are a set of usernames followed by the key username. Sets are chosen so that we have an efficient way to check if one user is following another. Examples are shown below.

```
# From users.csv
login_database = {
  "dr.b": "securepassword42",
  "tb12": "7rIng$",
  "hunter": "hunter2"
}

# From posts.csv
posts_database = {
  "dr.b": [
    {"timestamp": "2024-01-01 at 10:00", "title": "First Post", "content": "Coding
is fun!"},
    {"timestamp": "2024-01-02 at 9:52", "title": "Thought", "content": "New year's
was boring!"},
    {"timestamp": "2024-02-14 at 12:00", "title": "Holiday!", "content": "Happy
Valentine's Day!"}
  ],
  "tb12": [
    {"timestamp": "2024-01-11 at 11:00", "title": "Concussions", "content": "Drink
this fruit juice to prevent and cure concussion symptoms!"},
    {"timestamp": "2024-02-11 at 8:00", "title": "SB", "content": "The SB is not
the same without me."}
    ],
  "hunter": [{"timestamp": "2024-01-07 at 1:12", "title": "Test", "content": "This
is a test? I won't use this app."}]
}

# From follows.csv, tb12 follows nobody here.
follows_database = {
  "dr.b": {"tb12"},
  "hunter": {"dr.b", "tb12"}
}
```

Main Menu Specification

The main menu of the app has three options:

- Login
- Register
- Exit

Register Specification

The register screen has the following options:

- Register
- Back to Main Menu

The Register option asks the user for a username and password combination. If the username already exists, do not add the user to the `users.csv` file and tell the user that they must provide a unique username that does not contain spaces (this is so that a single user does not break the followers database. we use spaces to separate followed users). Provide an option to go back to the main menu of the app. If a user is successfully added to the database, redirect them to the main menu again. Update the `users.csv` file to reflect the newly added user.

Login Specification

The login screen redirects the user prompts the user for their username and password. If the username/password pair is valid, redirect the person to a user portal with the following options:

- Make Post
- Follow User
- View My Posts
- View Followed Posts
- Search User Posts
- Logout

The logout option will log the user out and redirect the user to the main menu.

The Make Post option will allow the user to make a post with a title and some content. This post will be saved in the `posts.csv` file.

Follow User prompts the user for another user to follow. It will update the `follows.csv` file if the followed user exists, and will not do anything if the followed user does not exist. Make sure that the user exists to avoid errors later.

The View My Posts option will display all of the posts for the current user sorted by most recently posted at the top.

The View Followed Posts option will display all of the posts by your followed users sorted as most recently posted at the top.

The Search User Posts will display posts by a specific user only, even if you are not following them.

Posts are displayed in the following format:

```
++++++++++++++++++++++++++++++++++++
==== POST TITLE ====
== posted by USERNAME at TIMESTAMP ==
POST CONTENT
++++++++++++++++++++++++++++++++++++
```