HW Music

(100 points)

Vocabulary:

- **Array** – An **array** is a series of memory locations, or 'boxes', each of which holds a single item of data. All data in an array must be of the same data type.
- **Procedure** – A **procedure** is a set of commands which can be executed in order. It is similar to a function except that it does not need to return anything.

A music library has been exported as a CSV file.
The file contains a list of *artist names*, *number of albums*, and *number of tracks*.
The file is sorted in ascending order of *artist name*.
The first ten rows of data from the file are shown below:

Artist Name	Number of Albums	Number of Tracks
2Pac	7	35
30 Seconds to Mars	1	1
50 Cent	9	46
Action Bronson	1	3
Aerosmith	4	4
Akon	4	4
Alanis Morrisette	1	1
Alexis Jordan	1	1
Alter Bridge	1	11
Amerie	1	1

Some of the main program code is shown below. It's not completely correct (for example, you are not making a function called `searchArtist`) but it will help you with parameter passing and identifying procedures and functions.

```
### MAIN ###
music = initialize()
music = readFile(music, <.csv>)
searchArtist(music, <desired artist>)
sortAlbums(music)
sortTracks(music)
```

What to submit:

Please submit the following on Canvas:

- A Python file called **hwMusic_final.py**. This file should contain:
  - All of the code described below
  - You can include #7 at the bottom of the file as a multiline string

Steps:

1. Create two global variables called `rows` and `cols`. Examine the contents music.csv and set the appropriate number of rows and columns. Then, write a function called `initialize` that initializes and returns an empty 2-D array of the appropriate size. This function should not take in any parameters. **(5 points)**

2. Create a function called `readFile` that reads the contents of the file music.csv, stores its data into the array and returns it. **(10 points)**
   Hint: You can start with the readFile function we made in lab, but it is not perfectly suited to this .csv file. It will also throw an error. You will need to do some research and make adjustments to the code.

3. Create a procedure called `printData` that prints the contents of the array in a user-friendly grid. You can call this procedure in the main program to check your outputs periodically. **(3 points)**

4. There are three parts to this problem.

   a. Create a function called `biSearchArtist` that uses the **binary search algorithm** to search the array for any artist and return the row number where it was found. **(15 points)**
   (This sample output is based on row numbers starting at 1; use it to check your work.)

   ```
   Nas was found at row number 197.
   ```

   b. Repeat part (a) using a **sequential search algorithm**. Call this function `seqSearchArtist`. **(10 points)**

c. Use the `timeit` module to compare each method. To do this, make two functions called `timeBiSearchArtist` and `timeSeqSearchArtist`. These functions should take no parameters, and they should return the time it takes to execute 100000 calls to the function. For grading purposes, please use 'Usher' as the artist. You will probably need to research how to do this – here is a link that may be helpful: https://www.guru99.com/timeit-python-examples.html **(10 points)**

5. There are three parts to this problem:

   a. Create a function called `bubbleSortAlbums` that uses the **bubble sort algorithm** to sort the array in descending order of *number of albums* and return it. **(12 points)**
   After running the function in your main program, call the `printData()` procedure to check the array (the first 10 rows are shown below).

Artist	Albums	Tracks
Eminem	13	93
Jay-Z	13	90
The Game	12	143
50 Cent	9	46
Kanye West	9	84
DMX	8	25
Ludacris	8	47
Nas	8	44
Snoop Dogg	8	31
Busta Rhymes	7	15

   b. Choose another sort method (either merge sort, selection sort, shell sort, or quick sort) to repeat your efforts in part (a). Include a docstring to indicate which method you chose. Call this function `altSortAlbums`. **(7 points)**

   c. Use the `timeit` module to compare each method. To do this, make two functions called `timeBubbleSortAlbums` and `timeAltSortAlbums`. These functions should take no parameters, and they should return the time it takes to execute 1000 calls to the function. Add some functionality in the main program that prints the results. **(3 points)**

6. There are three parts to this problem:

   a. Create a function called `inSortTracks` that uses the **insertion sort algorithm** to sort the array in descending order of *number of tracks* and return it. **(10 points)**
   After running the function in your main program, call the `printData()` procedure to check the array (the first 10 rows are shown below).

Artist	Albums	Tracks
Eminem	13	93
Jay-Z	13	90
Kanye West	9	84
Dr. Dre	7	57
Ludacris	8	47
50 Cent	9	46
Lil' Wayne	4	45
Nas	8	44
Young Jeezy	5	36
2Pac	7	35

   b. Choose another sort method (either merge sort, selection sort, shell sort, or quick sort) to repeat your efforts in part (a). Include a docstring to indicate which method you chose. **Do not use the same method you chose in (5b).** Call this function `altSortTracks`. **(7 points)**

   c. Use the `timeit` module to compare each method. To do this, make two functions called `timeInSortTracks` and `timeAltSortTracks`. These functions should take no parameters, and they should return the time it takes to execute 1000 calls to the function. Add some functionality in the main program that prints the results. **(3 points)**

7. Comment on your time analysis of your sort and search operations. Based on this project, can you say that a binary search is more efficient than a sequential search? Can you say that one sort is more efficient than another? Why or why not? **(5 points)**