DS 2nd exam test, February 8th, 2024

Download the data

1. Consider the file sciprog-ds-2024-02-08-FIRSTNAME-LASTNAME-ID.zip and extract it on your desktop.

2. Rename sciprog-ds-2024-02-08-FIRSTNAME-LASTNAME-ID folder:

   Replace **FIRSTNAME**, **LASTNAME**, and **ID** with your first name, last name and student id number. **Failure to comply with these instructions will result in the loss of 1 point on your grade.**

   like sciprog-ds-2024-02-08-alessandro-romanel-432432

   From now on, you will be editing the files in that folder.

3. Edit the files following the instructions.

4. At the end of the exam, **compress** the folder in a zip file

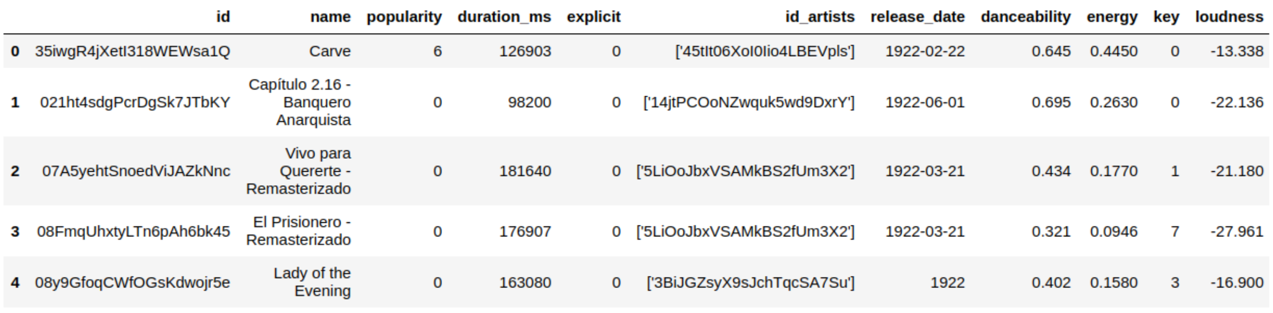
   sciprog-ds-2024-02-08-alessandro-romanel-432432.zip

   and submit it. This is what will be evaluated. Please, include in the zip archive all the files required to execute your implementations!

NOTE: You can only use the data structures and packages provided in the exam script files. **Importing other Python packages IS NOT allowed** unless explicitly stated in the exam instructions. Using Python collections or other libraries will impact your final grade. Still, **IT IS ALLOWED** to use **built-in Python operators** as we have done during the practical classes (max, min, len, reversed, list comprehensions, etc).

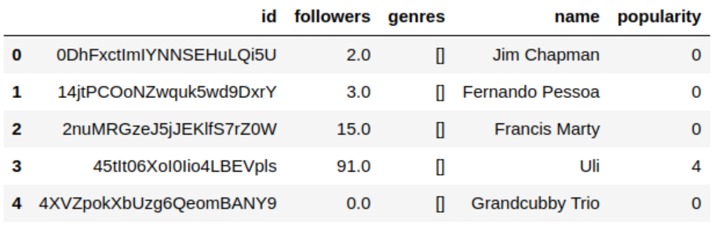
Exercise 1 [FIRST MODULE]

You have been provided with two CSV files: songs.csv and artists.csv, which are structured as follows:
songs:



	id	name	popularity	duration_ms	explicit	id_artists	release_date	danceability	energy	key	loudness
0	35iwgR4jXetI318WEWsa1Q	Carve	6	126903	0	['45tIt06XoI0Iio4LBEVpls']	1922-02-22	0.645	0.4450	0	-13.338
1	021ht4sdgPcrDgSk7JTbKY	Capítulo 2.16 - Banquero Anarquista	0	98200	0	['14jtPCOoNZwquk5wd9DxrY']	1922-06-01	0.695	0.2630	0	-22.136
2	07A5yehtSnoedViJAZkNnc	Vivo para Quererte - Remasterizado	0	181640	0	['5LiOoJbxVSAMkBS2fUm3X2']	1922-03-21	0.434	0.1770	1	-21.180
3	08FmqUhxtyLTn6pAh6bk45	El Prisionero - Remasterizado	0	176907	0	['5LiOoJbxVSAMkBS2fUm3X2']	1922-03-21	0.321	0.0946	7	-27.961
4	08y9GfoqCWfOGsKdwojr5e	Lady of the Evening	0	163080	0	['3BiJGZsyX9sJchTqcSA7Su']	1922	0.402	0.1580	3	-16.900

artists:



	id	followers	genres	name	popularity
0	0DhFxctImIYNNSEHuLQi5U	2.0	[]	Jim Chapman	0
1	14jtPCOoNZwquk5wd9DxrY	3.0	[]	Fernando Pessoa	0
2	2nuMRGzeJ5jJEKlfS7rZ0W	15.0	[]	Francis Marty	0
3	45tIt06XoI0Iio4LBEVpls	91.0	[]	Uli	4
4	4XVZpokXbUzg6QeomBANY9	0.0	[]	Grandcubby Trio	0

The songs file encompasses a compilation of songs played between 1922 and 1924, while the artists file comprises details about the artists.

Note that the entries in the "id_artists" column in the songs table correspond to those in the "id" column in the artists table.

1) load both the “songs.csv” and “artists.csv” files.

2) Print the song with the highest popularity by defining a function named highest_popularity. This function should accept a DataFrame as input and print both the name and the popularity of the song.

```
def highest_popularity(dataset):
        …
        print res

> highest_popularity(songs):
>> Title: Nobody Knows You When You're Down and Out
>>  Popularity: 41
```

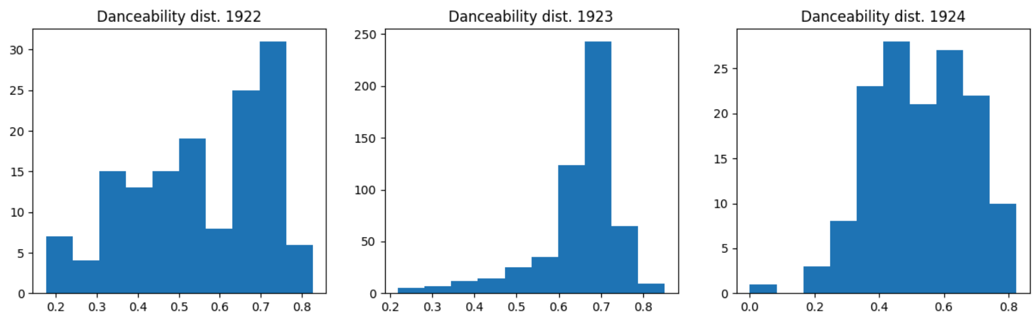
3) Create a function named longest_song_in_year that accepts the dataframe and a year as input (default value = 1923) and **returns** the longest song in that specific year. Ensure that the durations are presented in the format mm:ss:milliseconds.
   As an example, consider a duration of 416,984 milliseconds; the converted format would be 6 minutes, 56 seconds, and 984 milliseconds.

```
def longest_song_in_year(songs,year…):
        …
        return res

> longest_song_in_year(songs,1923)
>> ('Quiereme,Que ganas me dieron - en vivo', '6:56:984')
```

4) Create a function, named my_plot(), that generates a three-panel plot. Each panel should display the histogram of danceability for a specific year using plt.hist(). Ensure each plot has an appropriate title, and save the figure as "Danceability.pdf" for reference, similar to the provided figure.

```
def my_plot(songs):
        …
> my_plot(songs)
```



```
>
```

5) Conclusively, identify the artist ID associated with the highest number of songs played in a specified year. Subsequently, utilize the second file, "artists.csv," using the artist ID to retrieve the corresponding artist name. Define a function named most_frequent_artist that takes both DataFrames ("artists.csv" and "songs.csv") and a specific year as input. The function should **print** the artist's name, the given year, and the number of songs attributed to that artist.

```
def most_frequent_artist(songs,artists,year):
        …
        print

> most_frequent_artist(songs,artists,1924)
>> The artist: Francisco Canaro
>>  Played  59 songs in  1924
```

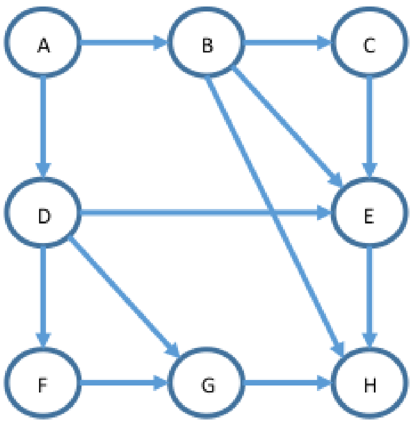
Exercise 2 [SECOND MODULE, theory]

Given a list *L* of *n* elements, please compute the asymptotic computational complexity of the following function, explaining your reasoning.

```
def func(L):
  n = len(L);
  k = 0;
  for i in range(n//2,n):
    j = 2;
    while j <= n:
      k = k + n // 2;
      j = j * 2;
  return k;
```

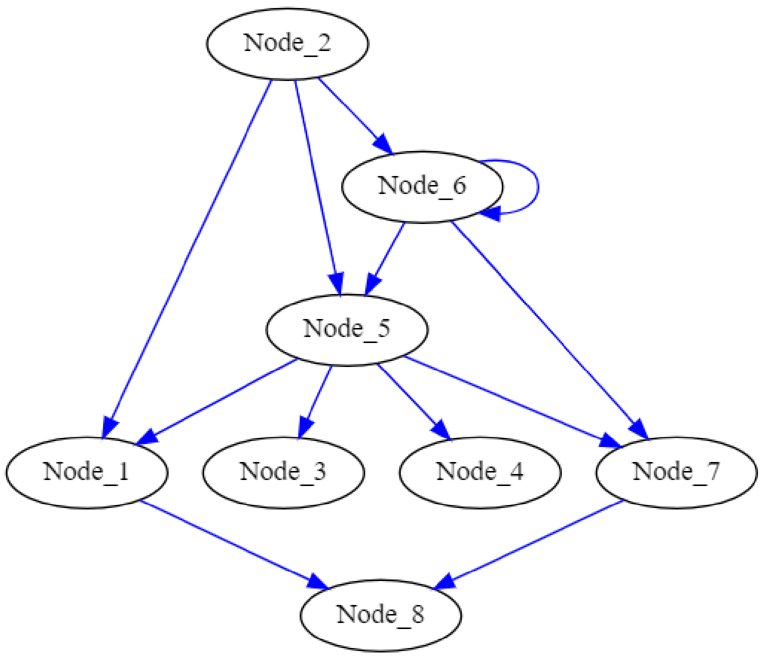
Exercise 3 [SECOND MODULE, theory]

What is the topological sorting of a directed acyclic graph (DAG)? Briefly describe an algorithm to compute it and provide a possible topological view of the following DAG.



Exercise 4 [SECOND MODULE, practical]

Consider the DiGraphLL class provided in exercise4.py implementing a directed graph by adjacency linked list. The graph created by the script looks like this:



You are asked to implement the following functions:

1) **`getInDegree(self, nodeA)`**

   This method first checks if the node is present in the graph. If it is not present, it throws an error. If present, this method **must** return the **number** (Integer) indicating the number of **INCOMING** edges into nodeA.

2) **`getOutDegree(self, nodeA)`**

   This method first checks if the node is present in the graph. If it is not present, it throws an error. If present, this method **must** return the **number** (Integer) indicating the number of **OUTGOING** edges from nodeA.

3) **`findMostConnectedNode(self)`**

   This method iterates over all the nodes of the graphs and calculates the degree for each node (the sum of incoming and outgoing edges). Then, it returns a tuple with the name of the most connected node (highest degree) and the degree of it (e.g. `("Node_5", 6))`