

# DS 1<sup>st</sup> exam test, January 23<sup>rd</sup>, 2025

## Download the data

1. Consider the file [sciprog-qcb-23-01-2025-FIRSTNAME-LASTNAME-ID.zip](#) and extract it on your desktop.
2. Rename [sciprog-qcb-23-01-2025-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-qcb-23-01-2025-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-qcb-23-01-2025-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	35iwgR4jXetl318WEWsa1Q	Carve	6	126903	0	[45tlt06Xol0lio4LBEVpls]	1922-02-22	0.645	0.4450	0	-13.338
1	021ht4sdgPcrDgSk7JTbKY	Capítulo 2.16 - Banquero Anarquista	0	98200	0	[14jtPCOoNZwqk5wd9DxrY]	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	08FmqUhxyLTn6pAh6bk45	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	14jtPCOoNZwqk5wd9DxrY	3.0	[]	Fernando Pessoa	0
2	2nuMRGzeJ5jJEKlfS7rZOW	15.0	[]	Francis Marty	0
3	45tlt06Xol0lio4LBEVpls	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.

---

### Exercise 1.1

Load the two `.csv` files.

---

### Exercise 1.2

Define a function named `max_popularity` that identifies the song with the highest popularity. The function should **print**:

1. The name of the song.
2. Its duration (in seconds).

```
def max_popularity(songs...  
    ....  
    print(....
```

```
> max_popularity(songs)
```

```
>> Nobody Knows You When You're Down and Out 177.133
```

---

### Exercise 1.3

Define a function named `max_popularity_with_duration` that takes the following inputs:

1. `songs` (a DataFrame containing song data),
2. `min_allowed_duration` (the minimum duration for filtering songs),
3. `max_allowed_duration` (the maximum duration for filtering songs).

The function should:

- Filter the songs to include only those with a duration within the specified range (`min_allowed_duration` to `max_allowed_duration`).
- Identify the song with the highest popularity among the filtered songs.
- **Return** the name of that song.

Set the default values for `max_allowed_duration` and `min_allowed_duration` to 4 minutes and 3 minutes, respectively.

```
def max_popularity_with_duration(...  
    ....  
    return
```

```
> max_popularity_with_duration(songs,2,3)  
>> Nobody Knows You When You're Down and Out
```

---

### Exercise 1.4

Determine if there is a correlation between the popularity of an artist and the popularity of their songs.

#### Instructions:

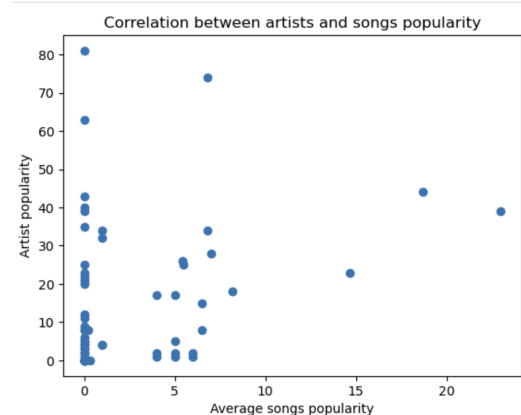
1. Write a function (named `popularity_correlation`) that takes as input the `artists` and `songs` DataFrames.
2. Within the function, create a dictionary where:
  - The **key** is the `artist_id`.
  - The **value** is a tuple containing:
    - The artist's popularity (from the `artists` DataFrame).
    - The average popularity of the songs associated with that artist (from the `songs` DataFrame).
3. Use this dictionary to create a scatter plot where:
  - The x-axis represents the artist's popularity.
  - The y-axis represents the average popularity of their songs.
4. Add the following to your scatter plot:

- Labels for the x-axis and y-axis.
- A descriptive title.

5. **Save the figure** as an image file. (see the figure below)

```
def popularity_correlation(...
```

```
....
    save
```



## Exercise 1.5

Define a function named `summary` that takes the following inputs:

1. `songs` (a DataFrame containing song data),
2. `artists` (a DataFrame containing artist data),
3. `acousticness_threshold` (a value used to filter songs by their acousticness level).

### Function Description:

1. Filter the `songs` DataFrame to include only songs with an acousticness level greater than the given `acousticness_threshold`.
2. For the filtered songs, calculate the following for each artist:
  - The artist's name.
  - The average duration (in seconds) of their songs.
  - The average acousticness threshold of their songs.
3. Write the results to a `.txt` file. Each line in the file should include:
  - The artist's name,
  - The average song duration,
  - The acousticness threshold. (see the figure below, and use the same format)

**Note:** Set the default value of `acousticness_threshold` to **0.995**.

```
def summary(...
```

```
....
    save
```

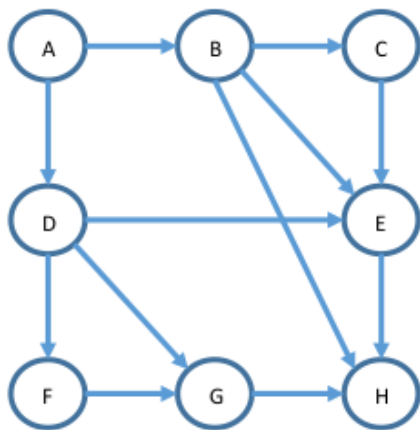
```
Acousticness Threshold: 0.995
Artist Name, Average Duration (seconds), Average Acousticness
Bessie Smith, 195.81, 0.996
Clarence Williams' Blue Five, 152.51, 0.996
Francisco Canaro, 168.45, 0.996
George Olsen, 186.33, 0.996
Ignacio Corsini, 157.8, 0.996
Jeanne Saint Bonnet, 150.85, 0.996
King Oliver's Creole Jazz Band, 170.83, 0.996
Louis Armstrong & His Hot Five, 161.8, 0.996
Cora Martin & Her Ten Band, 162.50, 0.996
```

### Exercise 2 [SECOND MODULE, theory]

Consider a hash table with separate chaining (implemented as mono-directional linked list) to handle collisions. The keys 14, 16, 4, 5, 35 and 18 are inserted into an initially empty hash table of length 7 using the hash function  $h(k) = (k+1)\%7$  (modulo operator). What is the resulting hash table?

### 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.



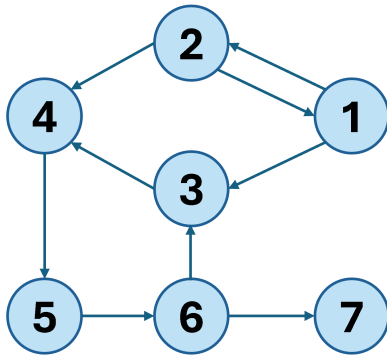
### Exercise 4 [SECOND MODULE, practical]

For this exercise, you need to use the provided `es4.py` script as a template. You should modify the script with the implementations requested below. Please note that:

- The code provided also contains the unittesting, comments were added to the code to highlight which parts you should **NOT** modify and which parts you are supposed to edit to include your implementations, as requested below.

Hints:

- the class `TestDiGraphAsAdjacencyMatrix` is the one you do not have to modify
  - the class `DiGraphAsAdjacencyMatrix` is the one you have to modify
  - you can uncomment the last 2 lines of the `__main__` function to test your implementations
- 
- The code provided will generate the following graph:



Given the direct graph class implemented using an adjacency matrix, as in the provided script (`es4.py`), **implement the following methods**:

1. Implement, as described in the Graph ADT, the standard `insertNode()` and `insertEdge()` functions
2. Implement, as described in the Graph ADT, the standard `deleteNode()` and `deleteEdge()` functions
3. Implement the method `BFS(start_node)` with the following requirements:
  - a. Takes as input the starting node label
  - b. Returns a list of labels of the visited nodes
  - c. Performs a check if the starting node is in the graph
  - d. Use the Python class `list()` data structure as a *queue* to keep track of the nodes to visit

Hints:

- The BFS visits the graph by levels (rows)
- Node labels are stored in a separate list inside the class

4. Implement the method `hasPath(start_node, end_node)` that given two nodes returns a boolean value if there exists a path between the two nodes.

Hints:

- Check that the two nodes are present in the graph
- You can exploit the previously implemented method