DS 2nd exam test, February 13th, 2023

Download the data

1. Consider the file sciprog-ds-2023-02-13-FIRSTNAME-LASTNAME-ID.zip and extract it on your desktop.

2. Rename sciprog-ds-2022-02-13-FIRSTNAME-LASTNAME-ID folder:

   put your name, lastname and id number

   like sciprog-ds-2023-02-13-luca-marchetti-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-2023-02-13-luca-marchetti-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!

Exercise 1 [FIRST MODULE]

You are given a dataset (Environment_Temperature.csv) containing the temperature for several countries. For each country, the dataset contains the average temperature in each month for the years 1961 to 2019.

	Area Code	Area	Months Code	Months	Element Code	Element	Unit	Y1961	Y1962	Y1963	...	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017	Y
0	2	Afghanistan	7001	January	7271	Temperature change	°C	0.777	0.062	2.744	...	3.601	1.179	-0.583	1.233	1.755	1.943	3.416	1.201	
1	2	Afghanistan	7002	February	7271	Temperature change	°C	-1.743	2.465	3.919	...	1.212	0.321	-3.201	1.494	-3.187	2.699	2.251	-0.323	
2	2	Afghanistan	7003	March	7271	Temperature change	°C	0.516	1.336	0.403	...	3.390	0.748	-0.527	2.246	-0.076	-0.497	2.296	0.834	
3	2	Afghanistan	7004	April	7271	Temperature change	°C	-1.709	0.117	0.919	...	2.591	1.712	1.417	-0.052	0.585	1.589	0.980	1.252	

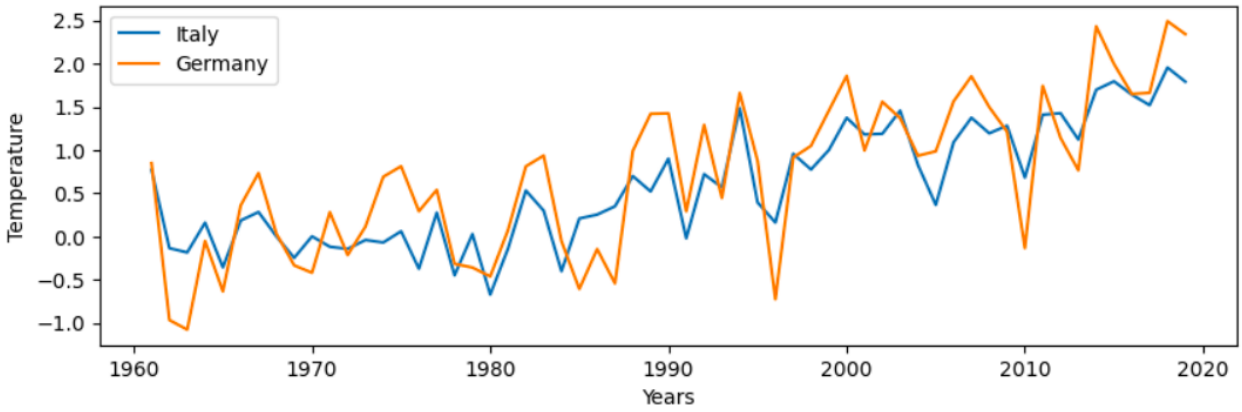
As you can see from the screenshot above, each row represents a specific country (Area Column) and a specific month (Months column). The average temperature is specified in different columns (i.e. the element in red represents the average temperature of January 2010 in Afghanistan).

1. Load the dataset “Environment_Temperature.csv”
2. Build a function (get_Area) that extracts all the rows of a specific country, then drop those columns having a "Months Code" greater than 7013.
   Use the function (get_Area), to build two new dataframe:
   -> *italy = get_Area(data,"Italy")*
   -> *germany = get_Area(data,"Germany")*

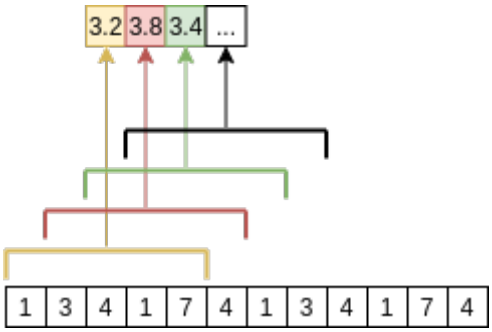
3. Buil a function (get_avg_temp) that given a year and the dataframe of a specific nation (i.e. italy) returns the average temperature of that year
   i.e
   ->get_avg_temp(italy,1961)
   -> 0.7703333333333333

4. Build a function (get_temperatures) that takes in input a dataframe of a specific nation (i.e. italy) iterates from 1961 to 2019 and computes the average temperature for each year (using the get_avg_temp() function previously defined).

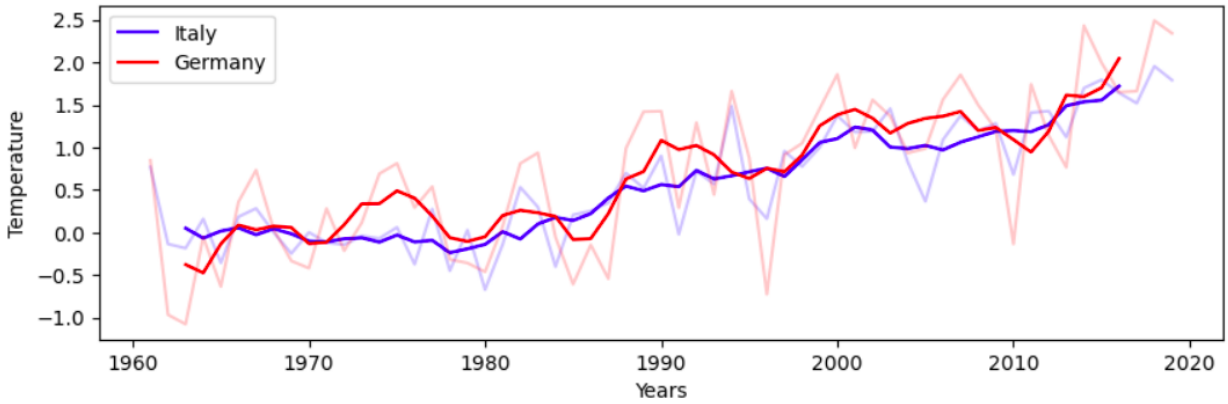
5. Plot the average temperatures from 1961 to 2020 in Italy and Germany in a unique plot. REMEMBER: set figsize=(10,3), xlabel = "Years", ylabel = "Temperature", the legend (Italy and Germany), and remember to add the correct year on the ylabel.



6. Improve the previous plot by adding a smooth version of the lines. To make a smooth version of the lines you should compute the average within a sliding window equal to 5. i.e. suppose the input array is [1,3,4,1,7,4,1,3, …], the new array will be [3.2, 3.8, 3.4 …] with a window of 5. Where 3.2 is the average of [1,3,4,1,7], 3.8 is the average of [3,4,1,7,4]. like the figure bellow



the new plot should be something like:



in which darker lines represent the smooth version, while lines with a smaller alpha (alpha = 0.2) are the original one.

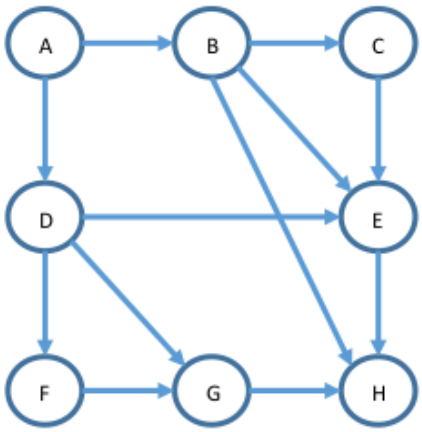
Exercise 2 [SECOND MODULE, theory]

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]

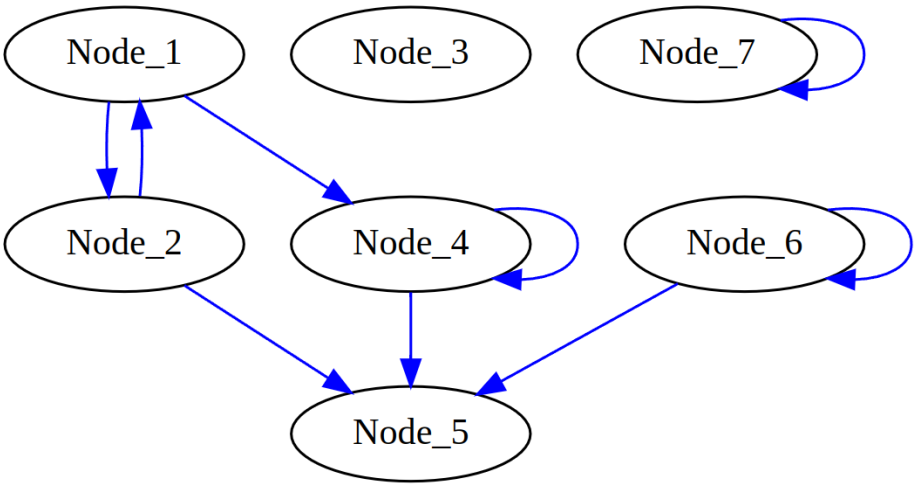
What is the topological sorting of a directed acyclic graph (DAG)? Briefly describe an algorithm to compute it and provide a possible topological view of the following DAG.



Exercise 4 [SECOND MODULE, practical]

Consider the *DiGraphLL* class provided in the file **exercise4.py** implementing a directed **graph** by **adjacency linked list**.

The graph structure is shown below:



Please **check carefully how the class is implemented** since it might slightly differ from the one you have seen during the practical class.

Implement the two missing methods:

1. *checkSelfEdge(self, node):*
   This method checks whether a node has a "self" edge, meaning an edge pointing to itself (e.g. node 4, 6,7). This method returns a boolean value.

   HINT: Remember to make sure that the node you are checking for a self edge is actually in the dictionary of nodes.

2. *isolatedNodes(self):*
   This method finds the nodes that are **isolated** from the graph (i.e. node 3, node 7), meaning those that do not have any incoming/outgoing edges. **Keep in mind** that a self-edge should not be considered as incoming/outgoing (i.e. Node 7 is considered **isolated**). This method returns a **list**.

   HINT: A *node A* **is** isolated only if:
   - its dictionary of edges (inner dict) does not contain other nodes except for itself or it's empty
   - *node A* is not present in the inner dict of **other** nodes of the graph (no incoming edges).