FINAL PROJECT

COMP202, Fall 2020

Due: Tuesday, December 22^{nd}, 11:59pm

Please read the entire PDF before starting. You must do this final project individually.

Question 1:	100 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment through automated tests.

To get full marks, you must:

- Follow all directions below.
  - In particular, make sure that all file names and function names are **spelled exactly** as described in this document. Otherwise, you will not be awarded points for those modules/functions.
- Make sure that your code runs.
  - Code with errors will receive a very low mark.
- Write your name and student ID as a comment at the top of all `.py` files you hand in.
- Name your variables appropriately.
  - The purpose of each variable should be obvious from the name.
- Comment your work.
  - A comment every line is not needed, but there should be enough comments to fully understand your program.
- Avoid writing repetitive code, but rather call helper functions! You are welcome to add additional functions if you think this can increase the readability of your code.
- Lines of code should NOT require the TA to scroll horizontally to read the whole thing. Vertical spacing is also important when writing code. Separate each block of code (also within a function) with an empty line.

The Final Project

In this project you will be analysing a data file containing recorded co2 emissions per country in the past 267 years. "Data cleaning" refers to the process of taking raw data and processing it into a state that can be used for empirical analysis. Your task in this project is to learn how to do a very simple clean of a data file and then use this data to perform some empirical analysis.

This project will allow you to better understand how to read and write to files, how to write objected oriented programs, and how to plot data using Matplotlib.

The project is also meant to raise awareness about the importance of a strong global response to climate change. The data shared with you in the large text file is **real data**, collected and share by the researchers at Our World in Data. Whether you are already a fan of Kurzgesagt or not, you might also enjoy this video. Though in this project we are not focusing on specific sources contributing to global warming, since COMP202 is an intro to computer programming course, we think it's important for us to mention the impact that the world's data centers are having on climate change. The infrastructure that allows us to "live on the cloud" has a carbon footprint higher than the aviation industry. Studies have found that the ICT's (Information and Communication Industry) contribution to global greenhouse gas emission could rise to 14% by 2040. The impact of our everyday usage of computers and technology might seem small at an individual level, but numbers add up fast. "As "Despacito" earned 5 billion streams on YouTube, it burned as much energy as 40,000 U.S. homes use in a year."[1] The fight against climate change starts with all of us. These holidays, let's all learn more about how to reduce our digital carbon footprint[2].

For full marks

Note that the project is designed for you to be practicing what you have learned in the videos up to and including Week 13. For this reason, you are NOT allowed to use anything seen after Week 13 (e.g., NumPy) or anything not seen in class at all. You will be heavily penalized if you do so.

In addition to the points listed on page 1, make sure to add the appropriate documentation string (docstring) to *all* the functions you write. The docstring must contain the following:

- The type contract of the function.
- A description of what the function is expected to do.
- At least 3 examples of calls to the function. You are allowed to use *at most* one example per function from this pdf.

Examples

For each question, we provide **examples** of how your code should behave. All examples are given as if you were to call the functions from the shell.

When you upload your code to codePost, some of these examples will be run automatically to check that your code outputs the same as given in the example. However, **it is your responsibility to make sure your code/functions work for any inputs, not just the ones shown in the examples.** When the time comes to grade your project, we will run additional, private tests that may use inputs not seen in the examples.

Furthermore, please note that your code files for this question and all others **should not contain any function calls in the main body of the program** (i.e., outside of any functions). Code that does not conform in this manner will automatically fail the tests on codePost and **be heavily penalized**. It is OK to place function calls in the main body of your code for testing purposes, but if you do so, make certain that you remove them before submitting. Please review what you have learned in video 5.2 if you'd like to add code to your modules which executes only when you run your files.

[1] https://twitter.com/fortunemagazine/status/1174979042920943616?lang=en
[2] https://climatecare.org/infographic-the-carbon-footprint-of-the-internet/

About the data

There are different files you'll need to work with during the assignment. The files containing data regarding the co2 emissions per countries will have the following information:

- A capitalized string representing the iso code of a given country. The ISO country codes are internationally recognized codes that designate every country.
- The name of the country
- A four digit integer representing the year to which the data record belongs.
- A number representing the co2 emissions in millions of tonnes for the specified country in the specified year. Note that some of the decimal numbers are recorded using a comma instead of a dot, which is what is used in French (and in Italian! ;)) for decimal numbers.
- An integer representing the population of the specified country in the specified year.

Here is how the first three lines of a file containing this data could look like:

```
QAT,Qatar,2001,41,215,615000
CMR-Cameroon-2001-3.324-16358000
COD Democratic Republic of Congo 2006 1,553 56578000
```

Safe Assumptions

You can assume the following:

- Every column will be present, even though some data might be missing (i.e., some columns may simply contain an empty string). The only data that can be missing in this manner is the number of co2 emissions or the population.
- There are no spelling mistakes
- The order in which the data appears follows the format described above.
- There are no entries (i.e., no rows) with the same country *and* the same year.

Part 1: Data clean up (20 points)

Create a module called `data_cleanup.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than doctest.

- Write a function called `find_delim`. The function takes a string as input representing a single line. A "delimiter" is a string that is used to separate columns of data on a single line. The function returns the most commonly used delimiter in the input string. The delimiters you should account for are tabs (`'\t'`), commas (`','`), spaces (`' '`), or dashes (`'-'`). You can assume that there won't be any ties for the most common delimiter. The function raises an `AssertionError` if there is no tab/comma/space/dash in the string. Don't worry that we have not seen `AssertionError` in class. We are deliberately using a different kind of error so that codePost can distinguish between the errors your code should raise and those that your code should not raise.

  For example,

  ```
  >>> find_delim("0 1 2 3,4")
  ' '
  >>> find_delim("cat\\tdog bat\\tcrab-cod")
  '\\t'
  ```

  Note that since the docstring is a string itself, for a tab character (or any escape sequence) to be evaluated as `'\t'`, we need to escape the backslash. This is why, **in the docstrings** tab characters will be represented as `'\\t'` instead of `'\t'`.

- Write a function called `clean_one`. The function takes as input two strings: the file name for a file to be read (`input_filename`), and file name for a file to be written (`output_filename`) respectively. The function will read the `input_filename`, make changes to each of the lines and write the new version to `output_filename`. Like for assignment 3, make sure to open all your files using the 'utf-8' encoding.

  The only change that should happen to the data is that in the output file all lines should have a tab as a delimiter in place of which ever delimiter each line originally had. The function should return an integer indicating the number of lines written to `output_filename`.

  For example:

  ```
  >>> clean_one('small_raw_co2_data.txt', 'small_tab_sep_co2_data.tsv')
  10

  >>> clean_one('large_raw_co2_data.txt', 'large_tab_sep_co2_data.tsv')
  17452
  ```

  Note that for the output file we are using the .tsv extension because the data is now all tab separated!

  Please check the provided files, *small_raw_co2_data.txt* and *small_tab_sep_co2_data.tsv*, for an example of how the data should change.

- Write a function called `final_clean`. The function takes as two strings: the file name for a file to be read (`input_filename`), and file name for a file to be written (`output_filename`) respectively. The function will read the `input_filename`, make changes to each of the lines and write the new version to `output_filename`. Like for assignment 3, make sure to open all your files using the 'utf-8' encoding. The input file has the same format as the one generated by the function `clean_one`.

  The function should change the data as follows:

  – All lines should have exactly 5 columns. From the changes that took place in the previous function we now might have lines with more than 5 columns. For examples, lines in which the co2 emissions were reported using commas and the delimiter was also a comma will now have

6 columns instead of 5. To find other cases in which the columns might have increased, please check the files provided and make sure to account for all of them. The only ways in which a line might end up with more than 5 columns are all found in the files provided.

- All commas which are used to indicate decimal numbers should be replaced with dots.

The function should return an integer indicating the number of lines written to `output_filename`.

For example,

```
>>> final_clean('small_tab_sep_co2_data.tsv', 'small_clean_co2_data.tsv')
10

>>> final_clean('large_tab_sep_co2_data.tsv', 'large_clean_co2_data.tsv')
17452
```

Please check the provided files, *small_tab_sep_co2_data.tsv* and *small_clean_co2_data.tsv*, for an example of how the data should change.

Part 2: Continents (15 points)

Create a module called `add_continents.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than doctest.

Note that a file named *iso_codes_by_continent.tsv* has been provided to you. This file contains several lines of data. Each line has the following contains the ISO country code and the continent to which each the country belongs, separated by a tab.

- Write a function called `get_iso_codes_by_continent` which takes as input a string representing a filename of a file that has the same format as *iso_codes_by_continent.tsv*. The function returns a dictionary mapping continents' names (all upper case) to a list of ISO codes (strings) of countries that belongs to that continent. The order in which the ISO codes appear in the list should be the same in which they appear in the input file.

  For example,

  ```
  >>> d = get_iso_codes_by_continent("iso_codes_by_continent.tsv")
  >>> len(d['ASIA'])
  50
  >>> len(d['NORTH AMERICA'])
  23
  >>> d['AFRICA'][0]
  'NGA'
  >>> d['EUROPE'][2]
  'BLR'
  ```

- Write a function called `add_continents_to_data` which takes as input three strings representing file names: `input_filename`, `continents_filename`, and `output_filename`. (Note that if you'd like, you can change the name of the input parameters.) The first file has the same format as the output file generated by the function `final_clean`. The second file has the same format as *iso_codes_by_continent.tsv* described above. The function will read the `input_filename`, make changes to each of the lines and write the new version to `output_filename`. Don't forget to open your files using the 'utf-8' encoding.

  The only change that should happen to the data is that in the output file a column should be added with the continent to which each country belongs. This should be the third column in the file, the one right after the name of the country. Note that there are some countries that are considered to be part of two continents. For these countries, write both continents separated by a comma.

The function should return an integer indicating the number of lines written to `output_filename`.

For example,

```
>>> add_continents_to_data("small_clean_co2_data.tsv", "iso_codes_by_continent.tsv",
"small_co2_data.tsv")
10

>>> add_continents_to_data("large_clean_co2_data.tsv", "iso_codes_by_continent.tsv",
"large_co2_data.tsv")
17452
```

Please check the provided files, *small_clean_co2_data.tsv* and *small_co2_data.tsv*, to see how the data should change. Note that if one of the lines in the input file were to be the following

```
RUS Russia 1971 1533.262 130831000
```

[UPDATED DEC 5: The line above mistankely contained a comma instead of a dot]

then the corresponding line in the output file would be written as follows:

```
RUS Russia ASIA,EUROPE 1971 1533.262 130831000
```

Please note that it does not matter if it seems to you that the number of spaces between each column is different. This is how tabs are represented.

Part 3: Countries (40 points)

Create a module called `build_countries.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than doctest and copy.

Create a class called `Country`. This class should have the following:

- Instance attributes: `iso_code` (a string), `name` (a string), `continents` (a list of strings), `co2_emissions` (a dictionary mapping integers to floats), `population` (a dictionary mapping integers to integers).
- Class attributes: `min_year_recorded` (an integer), `max_year_recorded` (an integer). These indicate the lowest and highest year (respectively) for which we have data recorded of all countries that have been created.
- A constructor that takes as input two strings (the iso code and the name of the country respectively), a list (the continents to which the country belongs), and integer (indicating the year in which the following data has been recorded), a float (indicating the co2 emissions of the country in the specified year in millions of tonnes), and a integer (indicating the population of the country in the specified year). The constructor uses these input to initialize all the instance attributes accordingly. Note that:
  - As for all functions/methods, you can assume that the type of the inputs will be correct
  - Valid ISO codes all contain 3 letters, beside the ISO code for Kosovo which is `'OWID_KOS'`. If the ISO code received as input is invalid, the constructor should raise an `AssertionError`.
  - The constructor should make a copy of the list received as input.
  - If the input representing the co2 emissions is a -1, this indicates that this data was not recorded for the specified year. In such case, the constructor will not add any items to the dictionary `co2_emissions`. Otherwise, the method will add a new key-value pair to the `co2_emissions` dictionary.

  - If the input representing the population is a -1, this indicated that this data was not recorded for the speficied year. In such case, the constructor will not add any items to the dictionary `population`. Otherwise, the method will add a new key-value pair to the `population` dictionary.
  - The constructor should update `min_year_recorded` and `max_year_recorded` if need be.

- A `__str__` method that returns a string representation of a country containing the name, the continents (separated by a comma if more than one), and a string representation of both the `co2_emissions` dictionary and the `population` dictionary. Each piece of information in the string should be separated by a tab.

  For example,

  ```
  >>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
  >>> str(r)
  'Russia\\tASIA,EUROPE\\t{2007: 1604.778}\\t{2007: 14266000}'
  ```

- An instance method called `add_yearly_data` which takes as input a string with the year, co2 emissions, and population, all separated by a tab. This method updates the appropriate attributes of the country. Note that if the co2 emission or the population data is an empty column, then no changes should be made to the corresponding attribute. Note also that this method should make sure to update `min_year_recorded` and `max_year_recorded` if need be.

  For example,

  ```
  >>> a = Country("AFG", "Afghnistan", ["ASIA"], 1949, 0.015, 7663783)
  >>> a.add_yearly_data("2018\\t9.439\\t37122000")
  >>> a.co2_emissions == {1949: 0.015, 2018: 9.439}
  True
  >>> a.population == {1949: 7663783, 2018: 37122000}
  True
  ```

- An instance method called `get_co2_emissions_by_year` which takes an integer as input. It returns the co2 emission of the country in the specified year if available. It returns 0.0 otherwise.

  For example,

  ```
  >>> a = Country("AFG", "Afghnistan", ["ASIA"], 1949, 0.015, 7663783)
  >>> a.add_yearly_data("2018\\t9.439\\t37122000")
  >>> a.get_co2_emissions_by_year(1949)
  0.015
  >>> a.get_co2_emissions_by_year(2000)
  0.0
  ```

- An instance method called `get_co2_per_capita_by_year` which takes an integer as input. It return the co2 emission per capita ***in tonnes*** (note that the co2 emissions for a country are recorded in millions of tonnes) for the specified year if available. If either the co2 emissions or the population of the country are not available for the specified year, the method returns `None`.

  For example,

  ```
  >>> a = Country("AFG", "Afghnistan", ["ASIA"], 1949, -1, 7663783)
  >>> a.add_yearly_data("2018\\t9.439\\t37122000")
  >>> round(a.get_co2_per_capita_by_year(2018), 5)
  0.25427
  >>> print(a.get_co2_per_capita_by_year(1949))
  None
  ```

- An instance method called `get_historical_co2` which takes an integer as input. It return the historical (total) co2 emission in millions of tonnes that the country has produced for all years up to and including the specified year.

  For example,

```
>>> q = Country("QAT", "Qatar", ["ASIA"], 2007, 62.899, 1218000)
>>> q.add_yearly_data("1993\\t30.985\\t501000")
>>> q.add_yearly_data("1989\\t14.292\\t462000")
>>> q.get_historical_co2(2000)
45.277
>>> q.get_historical_co2(2007)
108.176
```

- A class method called `get_country_from_data` which takes as input a string which has exactly the same format as the data stored in the output file generated by the function `add_continents_to_data`. The method should return a new Country object created from the data in the input string.

  For example,

```
>>> a = Country.get_country_from_data("ALB\\tAlbania\\tEUROPE\\t1991\\t4.283\\t3280000")
>>> a.__str__()
'Albania\\tEUROPE\\t{1991: 4.283}\\t{1991: 3280000}'
```

- A static method called `get_countries_by_continent` which takes as input a list of countries (i.e., objects of type Country). The method returns a dictionary mapping a string representing a continent to a list of countries (i.e., objects of type Country) which all belong to that continent. The order in which each country appears in the list should match the order in which they appeared in the input list.

  For example,

```
>>> a = Country("AFG", "Afghanistan", ["ASIA"], 1949, 0.015, 7663783)
>>> a.add_yearly_data("2018\\t9.439\\t37122000")
>>> b = Country("ALB", "Albania", ["EUROPE"], 2007, 3.924, 3034000)
>>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
>>> c = [a, b, r]
>>> d = Country.get_countries_by_continent(c)
>>> str(d['ASIA'][1])
'Russia\\tASIA,EUROPE\\t{2007: 1604.778}\\t{2007: 14266000}'
```

- A static method called `get_total_historical_co2_emissions` which takes as input a list of countries (i.e., objects of type Country) and an integer representing a year. The method returns a float representing the total co2 emissions (in millions of tonnes) produced by all the countries in the input list for all years up to and including the specified year.

  For example,

```
>>> b = Country("ALB", "Albania", ["EUROPE"], 2007, 3.924, 3034000)
>>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
>>> q = Country("QAT", "Qatar", ["ASIA"], 2007, 62.899, 1218000)
>>> b.add_yearly_data("1991\\t4.283\\t3280000")
>>> q.add_yearly_data("1993\\t30.985\\t501000")
>>> q.add_yearly_data("1989\\t14.292\\t462000")
>>> c = [b, r, q]
```

```
>>> Country.get_total_historical_co2_emissions(c,2007)
1721.161
>>> Country.get_total_historical_co2_emissions(c,2000)
49.56
```

- A static method called `get_total_co2_emissions_per_capita_by_year` which takes as input a list of countries (i.e., objects of type Country) and an integer representing a year. The method returns the co2 emissions per capita ***in tonnes*** produced by the countries in the given list in the specified year.

  If one of the two data point (co2 or population) is missing for a country in the list, then this country should be excluded when computing the value needed. More over, if the total co2 or the total population is 0, then the function should return 0.0.

  For example,

  ```
  >>> b = Country("ALB", "Albania", ["EUROPE"], 2007, 3.924, 3034000)
  >>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
  >>> c = [b, r]
  >>> round(Country.get_total_co2_emissions_per_capita_by_year(c,2007), 5)
  92.98855
  ```

- A static method called `get_co2_emissions_per_capita_by_year` which takes as input a list of countries (i.e., objects of type Country) and an integer representing a year. The method returns a dictionary mapping objects of type Country to floats representing the co2 emissions per capita ***in tonnes*** produced by the country in the specified year. Note that it is possible that some of the values in the output dictionary might be `None`. This could occur when the co2 per capita of that country for the specified year cannot be computed.

  For example,

  ```
  >>> b = Country("ALB", "Albania", ["EUROPE"], 2007, 3.924, 3034000)
  >>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
  >>> b.add_yearly_data("1991\\t4.283\\t3280000")
  >>> c = [b, r]
  >>> d1 = Country.get_co2_emissions_per_capita_by_year(c,2007)
  >>> len(d1)
  2
  >>> round(d1[r], 5)
  112.4897

  >>> d2 = Country.get_co2_emissions_per_capita_by_year(c, 1991)
  >>> print(d2[r])
  None
  >>> round(d2[b], 5)
  1.30579
  ```

- A static method called `get_historical_co2_emissions` which takes as input a list of countries (i.e., objects of type Country) and an integer representing a year. The method returns a dictionary mapping objects of type Country to floats representing the total co2 emissions (in millions of tonnes) produced by that country for all years up to and including the specified year.

  For example,

  ```
  >>> b = Country("ALB", "Albania", ["EUROPE"], 2007, 3.924, 3034000)
  >>> r = Country("RUS", "Russia", ["ASIA", "EUROPE"], 2007, 1604.778, 14266000)
  ```

```
>>> q = Country("QAT", "Qatar", ["ASIA"], 2007, 62.899, 1218000)
>>> b.add_yearly_data("1991\\t4.283\\t3280000")
>>> q.add_yearly_data("1993\\t30.985\\t501000")
>>> q.add_yearly_data("1989\\t14.292\\t462000")
>>> c = [b, r, q]
>>> d1 = Country.get_historical_co2_emissions(c,2007)
>>> len(d1)
3
>>> round(d1[q], 5)
108.176

>>> d2 = Country.get_historical_co2_emissions(c, 1991)
>>> print(d2[r])
0.0
>>> round(d2[b], 5)
4.283
```

- A static method called `get_top_n` which takes as input a dictionary mapping objects of type Country to numbers, and an integer `n`. The method returns a list of tuples. Each tuple is made up by the iso code of a country and the number to which the country is mapped in the input dictionary. Only the countries that map to the top `n` values should appear in the list. The tuples in the list should appear sorted on the values in descending order. If there are countries that map to the same values, the countries should be compared based on the alphabetical order of their *names*. Please note that this function should NOT modify the input dictionary.

  For example,

```
>>> a = Country("ALB", "Albania", [], 0, 0.0, 0)
>>> b = Country("AUT", "Austria", [], 0, 0.0, 0)
>>> c = Country("BEL", "Belgium", [], 0, 0.0, 0)
>>> d = Country("BOL", "Bolivia", [], 0, 0.0, 0)
>>> e = Country("BRA", "Brazil", [], 0, 0.0, 0)
>>> f = Country("IRL", "Ireland", [], 0, 0.0, 0)
>>> g = Country("MAR", "Marocco", [], 0, 0.0, 0)
>>> h = Country("NZL", "New Zealand", [], 0, 0.0, 0)
>>> i = Country("PRY", "Paraguay", [], 0, 0.0, 0)
>>> j = Country("PER", "Peru", [], 0, 0.0, 0)
>>> k = Country("SEN", "Senegal", [], 0, 0.0, 0)
>>> l = Country("THA", "Thailand", [], 0, 0.0, 0)
>>> d = {a: 5, b: 5, c: 3, d: 10, e: 3, f: 9, g: 7, h: 8, i: 7, j: 4, k: 6, l: 0}
>>> t = Country.get_top_n(d, 10)
>>> t[:5]
[('BOL', 10), ('IRL', 9), ('NZL', 8), ('MAR', 7), ('PRY', 7)]
>>> t[5:]
[('SEN', 6), ('ALB', 5), ('AUT', 5), ('PER', 4), ('BEL', 3)]
```

Finally, add to this module (but outside the class Country) a function called `get_countries_from_file`. This function takes as input a string representing a filename which has exactly the same format as the output file generated by the function `add_continents_to_data`. The function creates and return a dictionary mapping ISO country codes (strings) to objects of type Country based on the data in the file.

For example,

```
>>> d1 = get_countries_from_file("small_co2_data.tsv")
>>> len(d1)
9
>>> str(d1['ALB'])
'Albania\\tEUROPE\\t{2002: 3.748}\\t{2002: 3126000}'

>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> len(d2)
193
```

Part 4: Plotting data (25 points)

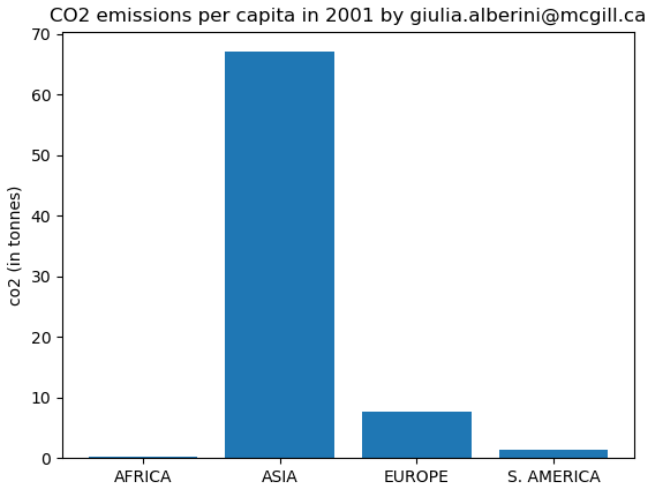
Create a module called `plot_data.py` and place your name and student ID at the top. All the functions in this section will go inside this module. You may not import any modules other than doctest, matplotlib, and the other modules you have created (if you need them for testing).

- Write a function called `get_bar_co2_pc_by_continent` which takes a dictionary as input like the one generated by `get_countries_from_file` and an integer representing a year. The function should create a bar plot representing the co2 emissions per capita (in tonnes) produced by all the countries in each continent. The bars should appear in alphabetical order, and the function should return a list of the values being plotted. The graph should have following:
  - The ylabel set as 'co2 (in tonnes)'
  - The title as 'CO2 emissions per capita in [year] by [your McGill email address]'. Of course, [year] should be replaced by the input integer and [your address] by your actual email.

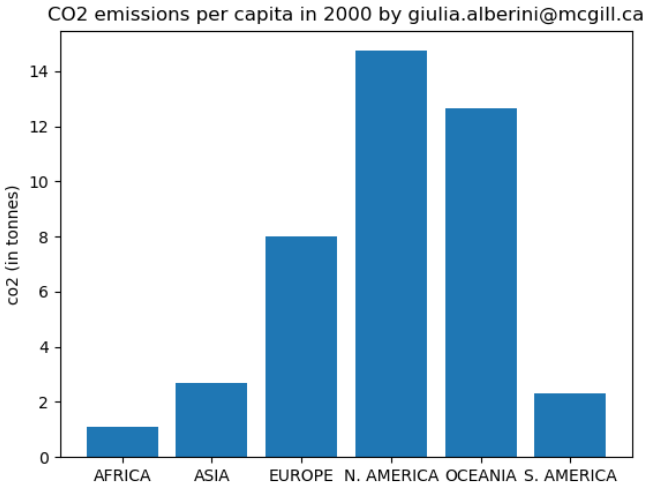
  The function should save the graph in a file named *co2_pc_by_continent_[year].png*, where year is replaced by the input integer.

  For example, the following instructions generate the graphs below.

```
>>> d1 = get_countries_from_file("small_co2_data.tsv")
>>> get_bar_co2_pc_by_continent(d1, 2001)
[0.20320332558992543, 67.01626016260163, 7.6609004739336495, 1.4196063588190764]
```



```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_bar_co2_pc_by_continent(d2, 2000)
>>> len(data)
6
>>> data[0] # AFRICA
1.0975340644568221
>>> data[3] # N. AMERICA
14.739682155717826
>>> round(data[4], 5) # OCEANIA
12.66302
```

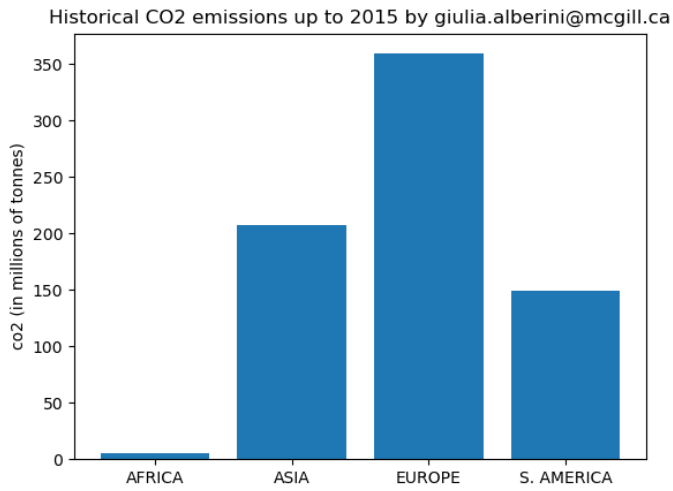


- Write a function called `get_bar_historical_co2_by_continent` which takes a dictionary as input like the one generated by `get_countries_from_file` and an integer representing a year. The function should create a bar plot representing the historical co2 emissions (in millions of tonnes) produced by all the countries in each continent. The bars should appear in alphabetical order, and the function should return a list of the values being plotted. The graph should have following:
  - The ylabel set as 'co2 (in millions of tonnes)'
  - The title as 'Historical CO2 emissions up to [year] by [your McGill email address]'. Of course, [year] should be replaced by the input integer and [your address] by your actual email.

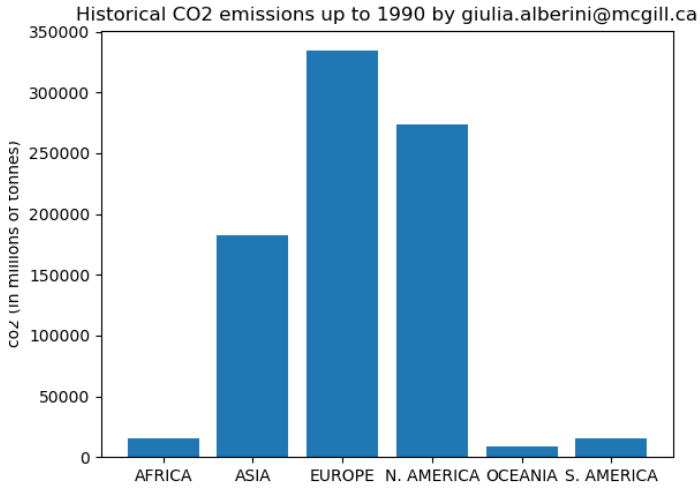
  The function should save the graph in a file named *hist_co2_by_continent_[year].png*, where year is replaced by the input integer.

  For example, the following instructions generate the graphs below.

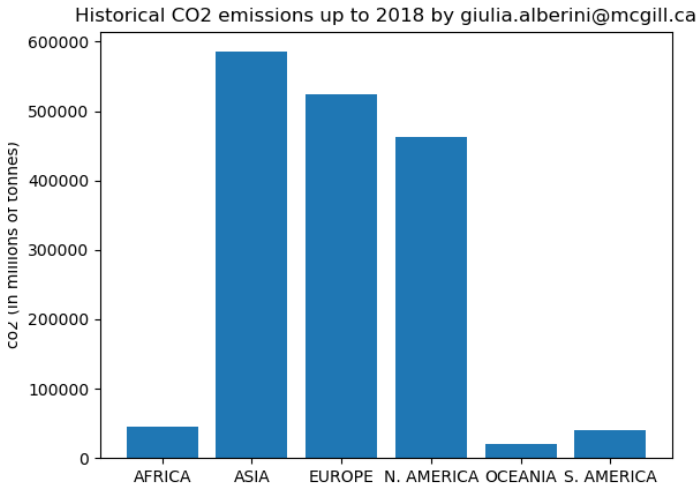
```
>>> d1 = get_countries_from_file("small_co2_data.tsv")
>>> get_bar_historical_co2_by_continent(d1, 2015)
[4.877, 207.54500000000002, 359.367, 149.34300000000002]
```




```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_bar_historical_co2_by_continent(d2, 1990)
>>> len(data)
6
>>> round(data[2],4) # EUROPE
334210.701
>>> data[4] # OCEANIA
8488.463
```




```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_bar_historical_co2_by_continent(d2, 2018)
>>> len(data)
6
>>> round(data[1], 5) # ASIA
585465.903
>>> round(data[4], 5) # OCEANIA
19845.01
```

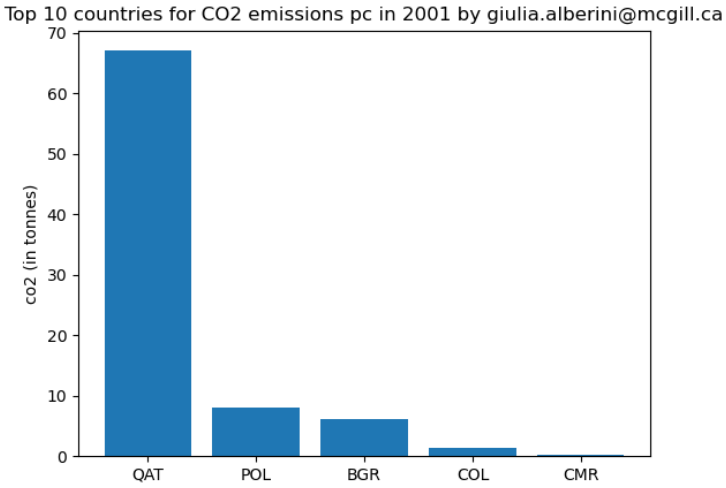


- Write a function called `get_bar_co2_pc_top_ten` which takes a dictionary as input like the one generated by `get_countries_from_file` and an integer representing a year. The function should create a bar plot representing the co2 emissions per capita (in tonnes) produced by the top 10 producing countries in the dictionary (if the dictionary contains less than 10 countries, then you should graph all of them). The bars should appear in order of co2 produced, and the function should return a list of the values being plotted. The graph should have following:
  - The ylabel set as 'co2 (in tonnes)'
  - The title as 'Top 10 countries for CO2 emissions pc in [year] by [your McGill email address]'. Of course, [year] should be replaced by the input integer and [your address] by your actual email.

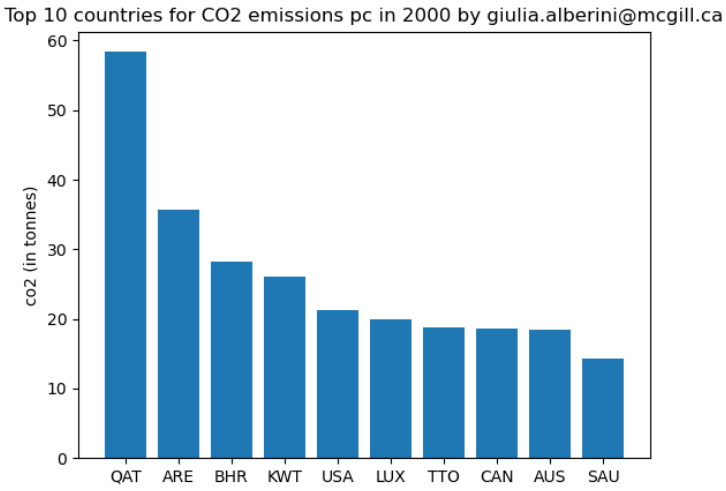
  The function should save the graph in a file named *top_10_co2_pc_[year].png*, where year is replaced by the input integer.

  For example, the following instructions generate the graphs below.

```
>>> d1 = get_countries_from_file("small_co2_data.tsv")
>>> data = get_bar_co2_pc_top_ten(d1, 2001)
>>> len(data)
5
>>> data[0]
67.01626016260163
>>> data[4]
0.20320332558992543
```




```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_bar_co2_pc_top_ten(d2, 2000)
>>> len(data)
10
>>> data[0] # QAT
58.388513513513516
>>> data[4] # USA
21.288834407209247
>>> data[9] # SAU
14.341511807975223
```



- Write a function called `get_bar_top_ten_historical_co2` which takes a dictionary as input like the one generated by `get_countries_from_file` and an integer representing a year. The function should create a bar plot representing the historical co2 emissions (in millions of tonnes) produced by the top 10 producing countries in the dictionary (if the dictionary contains less than 10 countries, then you should graph all of them). The bars should appear in order of co2 produced, and the function should return a list with the values being plotted. The graph should have following:

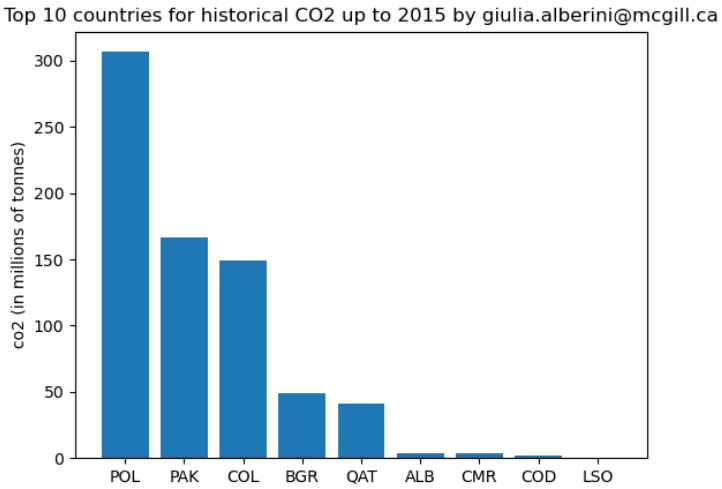
– The ylabel set as 'co2 (in millions of tonnes)'

– The title as 'Top 10 countries for historical CO2 up to [year] by [your McGill email address]'. Of course, [year] should be replaced by the input integer and [your address] by your actual email.

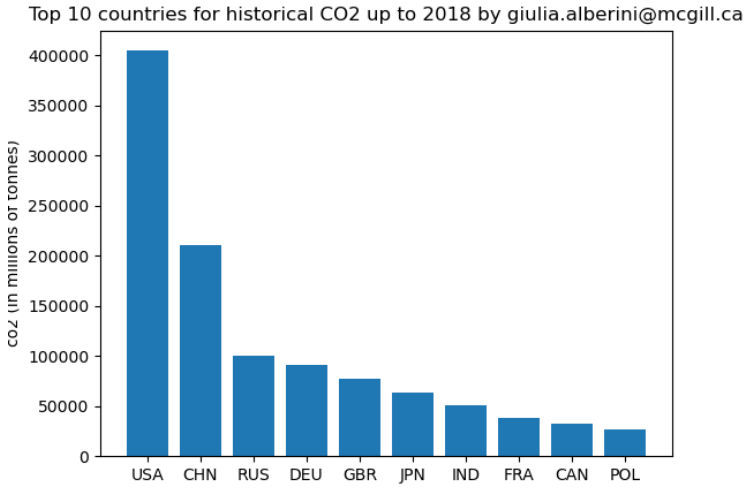
The function should save the graph in a file named *top_10_hist_co2_[year].png*, where year is replaced by the input integer.

For example, the following instructions generate the graphs below.

```
>>> d1 = get_countries_from_file("small_co2_data.tsv")
>>> get_bar_top_ten_historical_co2(d1, 2015)
[306.696, 166.33, 149.34300000000002, 48.923, 41.215, 3.748, 3.324, 1.553, 0.0]
```




```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_bar_top_ten_historical_co2(d2, 2018)
>>> len(data)
10
>>> data[0] # USA
404769.397
>>> data[1] # CHN
210201.179
>>> data[8] # CAN
32517.775
```



- Write a function called `get_plot_co2_emissions` which takes as input the following: a dictionary like the one generated by `get_countries_from_file`, a list of strings representing ISO codes, an integer `min_year`, and another integer `max_year`. You can assume that the list does not have more than 5 elements. The function should plot the co2 emissions of the selected countries (those whose ISO code appears in the input list) from `min_year` to `max_year`. You should use a different style for each line plotted (it's up to you to choose the style), and you should plot a maximum of 10-11 data points (to keep the plot readable). To do that, plot the data of the years from `min_year` to `max_year` using a step obtained by taking the number of total years and dividing it by 10. The function should return a 2D list. Each sublist should contain the co2 emission of a selected country from `min_year` to `max_year`. The position of the sublists should match the position of the ISO code in the input list.

  The graph should have following:

  - The ylabel set as 'co2 (in millions of tonnes)'
  - The title as 'CO2 emissions between [min_year] and [max_year] by [your McGill email address]'. [min_year] and [max_year] should be replaced by the input integers and [your address] by your actual email.
  - A legend should appear in the graph with the ISO codes of the countries being plotted. You can do this with:

    ```
    plt.legend(iso_codes)
    ```

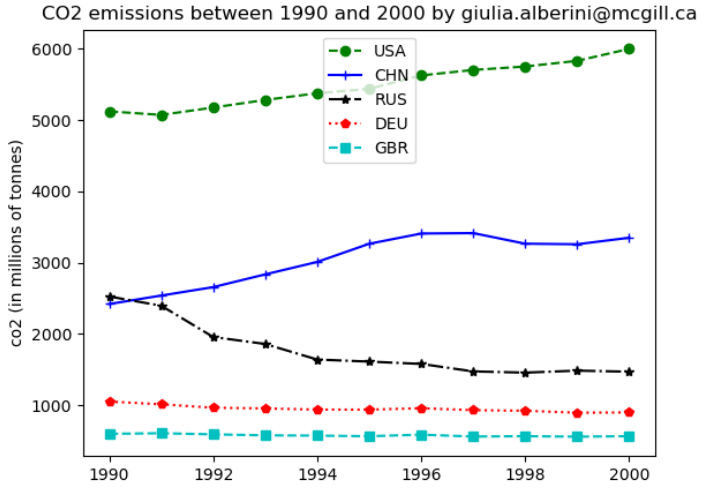
    where `iso_codes` is the list received as input.

  The function should save the graph in a file named *co2_emissions_[min_year]_[max_year].png*, where [min_year] and [max_year] are replaced by the input integers.

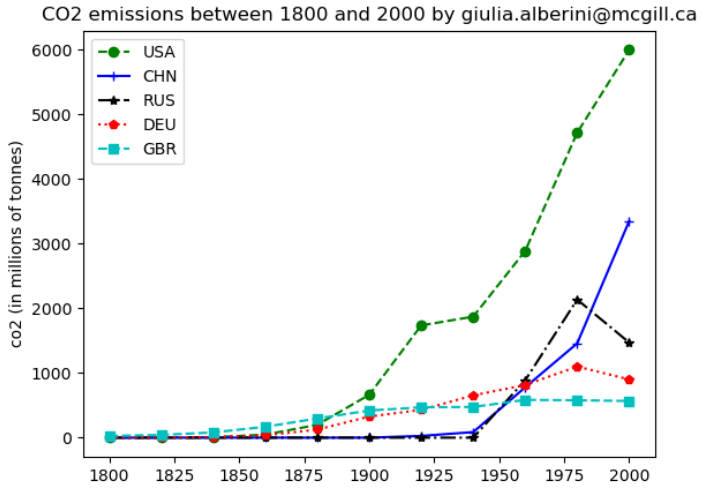
  For example,

  ```
  >>> d2 = get_countries_from_file("large_co2_data.tsv")
  >>> data = get_plot_co2_emissions(d2, ["USA", "CHN", "RUS", "DEU", "GBR"], 1990, 2000)
  >>> len(data)
  5
  >>> len(data[1]) # CHN
  11
  >>> data[0][:5] # USA
  ```

```
[5121.179, 5071.564, 5174.671, 5281.387, 5375.034]
```




```
>>> d2 = get_countries_from_file("large_co2_data.tsv")
>>> data = get_plot_co2_emissions(d2, ["USA", "CHN", "RUS", "DEU", "GBR"], 1800, 2000)
>>> len(data[0]) # USA
201
>>> data[2][4] # RUS
0.0
>>> data[4][190] # GBR
600.773
>>> data[3][200] # DEU
900.376
```



What To Submit

You must submit all your files on codePost (https://codepost.io/). The file you should submit are listed below. Any deviation from these requirements may lead to lost marks.

`data_cleanup.py`
`add_continents.py`
`build_countries.py`
`plot_data.py`
`README.txt` In this file, you can tell the TA about any issues you ran into doing this assignment. Remember that this assignment like all others is an `individual` assignment and must represent the entirety of your own work. You are permitted to verbally discuss it with your peers, as long as no written notes are taken. If you do discuss it with anyone, please make note of those people in this `README.txt` file. If you didn't talk to anybody nor have anything you want to tell the TA, just say "nothing to report" in the file.