As the second part of our final project, you will improve the quality of the world map plots that you created using pygal in last week's project. This improvement will rely on creating a better mapping from pygal country codes to World Bank country names. When complete, your code should have graphical functionality similar to this page at the World Bank data site. The primary goal of this assignment is to gain more hands-on experience working with multiple dictionaries as well as using multiple data sources to reconciling conflicting information . The assignment will also expose the student to some typical issues that arise in cleaning and unifying multiple sets of data.

Preliminaries: Working on the Project

As with the previous project, you should follow the course coding style guidelines. We have also provided you with an Owltest page to receive a preliminary grade and feedback on your project. Remember that each OwlTest link is specific to a particular project and that it does not submit a grade to Coursera.

When you are ready to submit your code to be graded formally, submit your code to the assignment page for this project. You will be prompted to open a tool which will take you to the Coursera LTITest page. Note that the Coursera LTITest page looks similar to the OwlTest page, but they are *not* the same! The CourseraLTI Test page has a **white** background and does submit your grade to Coursera.

Project: Plotting GDP Data on a World Map - Part 2

In this project, our fundamental task is to connect Pygal country codes to World Bank GDP country data. For last week's part of the project, our approach was to link Pygal's map dictionary (mapping 2-letter country codes to country names) to a dictionary that stored World Bank GDP country data keyed by the `"Country Name"` field.

This approach worked modestly well, but left part of the resulting world map blank with no plotted GDP data. In many of these cases, the missing data was due to slight differences in how a country's name was represented in Pygal vs. the World Bank data set. For example, the country name `"Viet Nam"` in pygal corresponds to the country name `"Vietnam"` in the World Bank data.

An alternative approach would be to use the three-letter `"Country Code"` in the World Bank data set to reconcile Pygal's country information with the World Bank's country information. In this case, we will need to determine how to map Pygal's two-letter country codes (e.g. `"vn"` for Vietnam) to the World Bank's three-letter country codes (e.g `"VNM"` for Vietnam).

Provided Data

We will again use the World Bank data from the last week's project for this two-part project. Our version of this data is available here as a CSV file. **Please use this version for your project** since having everyone work from the same data set will make testing your code much easier.

The Open Knowledge Network maintains a comprehensive collection of data on country codes. This data is organized in a CSV file that we have downloaded and slightly edited for consistency. Our version is available here. (We have deleted the second and third lines of the OKFN's CSV file.) Again, **please use this version for your project** since having everyone work from the same data set will make testing your code much easier.

We have provided the following template that you can use to get you started on the project. It includes the signatures (name, parameters, and docstrings) for all of the functions that you will need to write. The code however, simply returns some arbitrary value no matter what the inputs are, so you will need to modify the body of the function to work correctly. You should not change the signature of any of the functions in the template, but you may add any code that you need to.

Working with the CSV Files

As with the last project, we will again use a gdpinfo dictionary to specify information about the GDP data file. The gdpinfo dictionary is exactly the same as before and contains the following keys, all of which are strings (the use of these keys will become apparent as you work on the project, you may want to refer back to this information as you work on the different parts of the project):

- "gdpfile": the name of the CSV file that contains GDP data.
- "separator": the delimiter character used in the CSV file.
- "quote": the quote character used in the CSV file.
- "min_year": the oldest year for which there is data in the CSV file.
- "max_year": the most recent year for which there is data in the CSV file.
- "country_name": the name of the column header for the country names.
- "country_code": the name of the column header for the country codes.

If you look in the template file, you will see an example of such an "gdpinfo" dictionary that is used to access the GDP data from the World Bank discussed above.

As the format of the CSV file that stores the country codes could change (or you could acquire data from somewhere else), the functions that operate directly on the country code data will all take a "codeinfo" dictionary that provides information about the file. That way, you do not need to use constants within your code to access the CSV file and their columns. The codeinfo dictionary contains the following keys, all of which are strings (the use of these keys will become apparent as you work on the project, you may want to refer back to this information as you work on the different parts of the project):

- "codefile": the name of the CSV file that contains country codes.
- "separator": the delimiter character used in the CSV file.
- "quote": the quote character used in the CSV file.
- "plot_codes": the name of the column header that holds the country codes used by the plot library.
- "data_codes": the name of the column header that holds the country codes used by the GDP data.

If you look in the template file, you will see an example of such an "codeinfo" dictionary that is used to access the country codes from the Open Knowledge Network discussed above. It looks as follows:

```
codeinfo = {
    "codefile": "isp_country_codes.csv", # Name of the country code CSV file
    "separator": ",",                    # Separator character in CSV file
    "quote": '"',                        # Quote character in CSV file
    "plot_codes": "ISO3166-1-Alpha-2",   # Plot code field name
    "data_codes": "ISO3166-1-Alpha-3"    # GDP data code field name
}
```

While we have provide real GDP data, we strongly recommend you write smaller tests and utilize OwlTest to test each function you write. If something goes wrong, you will likely want to write smaller tests to help you understand how your code is working anyway. OwlTest uses smaller files to allow more targeted and understandable testing. OwlTest also uses smaller country code conversion files. You can use those files on your own, as well. Once you have everything working, you should be able to operate on the real data.

Problem 1: Generate a dictionary that maps different country codes to each other

The main task of this project is to process the World Bank GDP data and build a Pygal map dictionary whose values represented the GDP data for a given year. In this week's version of the project, the key step in this process is reconciling Pygal country codes/names with the World Bank country codes. In order to do this, you will first write a function called **`build_country_code_converter`** that takes **`codeinfo`**, a country code info dictionary, and returns a dictionary that maps plot country codes to data country codes.

Here is the signature of the **`build_country_code_converter`** function:

```
def build_country_code_converter(codeinfo):
    """
    Inputs:
      codeinfo      - A country code information dictionary

    Output:
      A dictionary whose keys are plot country codes and values
      are world bank country codes, where the code fields in the
      code file are specified in codeinfo.
    """
```

You need to write the code that implements this function.

Hints:

1. You should be using functions that you have already written in your implementation of this function. In particular, remember that you have already written a function to read a CSV file for the previous projects.
2. Remember that the **`codeinfo`** dictionary specifies which columns from the CSV file you should be using to create the dictionary. This will allow your program to continue to work if Pygal changes what country codes they use, you get your GDP data from a different source, and so on.

Problem 2: Create a dictionary that maps Pygal country codes to World Bank country codes

Next, you first must write a function called **`reconcile_countries_by_code`** which takes **`codeinfo`**, a country code information dictionary, **`plot_countries`**, a dictionary mapping country codes used by a plot library (such as Pygal) to the corresponding country name, and **`gdp_countries`**, a dictionary whose keys are the country codes used within the GDP data. The values in the **`gdp_countries`** dictionary are irrelevant for this function, but presumably contain GDP data for each country. The **`reconcile_countries_by_code`** function should return a dictionary and a set. The dictionary should map the country codes from **`plot_countries`** to country codes from **`gdp_countries`** that correspond to the same country (given the country code equivalencies specified by **`codeinfo`**). It should not contain key-value pairs for the countries within **`plot_countries`** that do not appear in **`gdp_countries`**. The set should contain all of the country codes within **`plot_countries`** that did not match any countries in **`gdp_countries`**, so is effectively the set of countries that the plot library knows about but cannot be found within the GDP data.

Here is the signature of the **`reconcile_countries_by_code`** function:

```
def reconcile_countries_by_code(codeinfo, plot_countries, gdp_countries):
    """
    Inputs:
      codeinfo       - A country code information dictionary
      plot_countries - Dictionary whose keys are plot library country codes
                       and values are the corresponding country name
      gdp_countries  - Dictionary whose keys are country codes used in GDP data

    Output:
      A tuple containing a dictionary and a set.  The dictionary maps
      country codes from plot_countries to country codes from
      gdp_countries.  The set contains the country codes from
      plot_countries that did not have a country with a corresponding
      code in gdp_countries.

      Note that all codes should be compared in a case-insensitive
      way.  However, the returned dictionary and set should include
      the codes with the exact same case as they have in
      plot_countries and gdp_countries.
    """
```

You need to write the code that implements this function.

Hints:

1. The country codes within the CSV file specified by **`codeinfo`** may not be in the same case as those within **`plot_countries`** and **`gdp_countries`**. You will therefore need to come up with a way to compare country codes while ignoring case. In other words, you should consider "abc" and "aBC" to be a match. However, make sure that the output dictionary contains the codes exactly as they appear in **`plot_countries`** and **`gdp_countries`**.
2. Be sure to look at the contents of the returned set. This should give you an understanding of how well the two sets of country codes agree.
3. If you code runs correctly, using the Pygal countries and the World Bank data, there should be 8 countries that are not found in the World Bank data. Examine these missing country codes and analyze why the mapping still fails. Note that in some case (*e.g.* Antartica), the lack of any World Bank data is understandable.

Problem 3: Transform GDP data for given year into a form suitable for a world map plot

Your next task is to implement the main function that processes GDP data. You must write a function called **`build_map_dict_by_code`** which takes **`gdpinfo`**, a GDP information dictionary (as used in the previous projects), **`codeinfo`**, a country code information dictionary, **`plot_countries`**, a dictionary mapping country codes used by a plot library (such as Pygal) to the corresponding country name, and **`year`**, the year for which to create a GDP map dictionary, expressed as a string. The **`build_map_dict_by_code`** function should return a dictionary and two sets. The dictionary should map the country codes from **`plot_countries`** to the log (base 10) of the GDP for the associated country in the given year. (The

logarithmic scaling is chosen to yield a better distribution of color shades in the final plot.) The first set should contain the country codes from **plot_countries** that were not found in the GDP data file. The second set contains the country codes from **plot_countries** that were found in the GDP data file, but have no GDP information for the specified year.

Here is the signature of the **build_map_dict_by_code** function:

```
def build_map_dict_by_code(gdpinfo, codeinfo, plot_countries, year):
    """
    Inputs:
      gdpinfo        - A GDP information dictionary
      codeinfo       - A country code information dictionary
      plot_countries - Dictionary mapping plot library country codes to country
        names
      year           - String year for which to create GDP mapping

    Output:
      A tuple containing a dictionary and two sets.  The dictionary
      maps country codes from plot_countries to the log (base 10) of
      the GDP value for that country in the specified year.  The first
      set contains the country codes from plot_countries that were not
      found in the GDP data file.  The second set contains the country
      codes from plot_countries that were found in the GDP data file, but
      have no GDP data for the specified year.
    """
```

You need to write the code that implements this function.

Hints:

1. Your implementation of **build_map_dict_by_code** should be similar to your implementation of **build_map_dict_by_name** from the previous project.
2. Think about what key you need to use in order to create an appropriate dictionary mapping the GDP data in order to use the other functions you've written for this project.

Problem 4: Create an SVG image of the GDP data plotted on the world map

As the final part of this project, your task is to write a function that takes the GDP map information computed using **build_map_dict_by_code** and create a world map plot using Pygal. You should write a function called **render_world_map** which takes **gdpinfo**, a GDP information dictionary, **codeinfo**, a country code information dictionary, **plot_countries**, a dictionary mapping country codes used by Pygal to the corresponding country name, and **year**, the string year for which to create a GDP map dictionary, and **map_file**, the string name of the file to write the output plot to.

Using these inputs, you should use Pygal to plot the logarithmically-scaled GDP data on a world map. Review Pygal's documentation on world maps for more details. Make sure that you plot not only the GDP data, but also the countries which are missing from the GDP data entirely and the countries that are contained within the GDP data, but have no data for the given year.

The output plot should be stored in an SVG file with the name specified by the **`map_file`** input.

Here is the signature of the **`render_world_map`** function:

```
def render_world_map(gdpinfo, codeinfo, plot_countries, year, map_file):
    """
    Inputs:
      gdpinfo        - A GDP information dictionary
      codeinfo       - A country code information dictionary
      plot_countries - Dictionary mapping plot library country codes to country
        names
      year           - String year of data
      map_file       - String that is the output map file name

    Output:
      Returns None.

    Action:
      Creates a world map plot of the GDP data in gdp_mapping and outputs
      it to a file named by svg_filename.
    """
```

You need to write the code that implements this function.

Hints:

1. Your implementation of **`render_world_map`** should be nearly identical to the last project's version.
2. As was the case for last week's project, you can also render the output directly to your browser using the **`render_in_browser()`** method.
3. The precise labels on your plots are up to you. As a general rule of thumb, your plot should have an informative title and legend entries. See Pygal's documentation on how to add such labels to your plot.
4. We suggest you compare your maps from this project to the last project. If everything is working correctly, there should be fewer countries "missing".

VERY IMPORTANT FINAL NOTE: OwlTest does not include tests for the final function **`render_world_map`** since specifying the precise format of the resulting output image is extremely difficult. Therefore, your grade for this project will depend on the correctness of your implementation of the other required functions. You are welcome to self-assess the correctness of your

version of the final function **`render_world_map`** based on the sample output images provided below (note that the function **`test_render_world_map`** in the template will call your **`render_world_map`** function appropriately to create these images):

- **`year = "1960"`** produces the world map plot isp_gdp_world_code_1960.svg.
- **`year = "1980"`** produces the world map plot isp_gdp_world_code_1980.svg.
- **`year = "2000"`** produces the world map plot isp_gdp_world_code_2000.svg.
- **`year = "2010"`** produces the world map plot isp_gdp_world_code_2010.svg.

Mark as completed