As you grow as a scripter, you will learn to use packages created by others as part of your scripts. In the final three required projects of this specialization, you will work with the Python visualization package Pygal. In these assignments, you will learn to process data stored in CSV form using dictionaries and create plots of this data using Pygal.

Preliminaries: Working on the Project

Coding Style

In this class, you will be asked to strictly follow a set of coding style guidelines. Good programmers not only get their code to work, but they also write it in a way that enables others to easily read and understand their code. Please read the style guidelines carefully and get into the habit of following them right from the start. A portion of your grade on the project will be based upon coding style.

Testing

You should always test your code as you write it. Do not try to solve the entire project before running it! If you do this, you will have lots of errors that interact in unexpected ways making your program very hard to debug. Instead, as you write each function, make sure you test it to ensure that it is working properly before moving on to the next function.

Throughout this course, we will be using a machine grader (OwlTest) to help you assess your code. You can submit your code to this Owltest page to receive a preliminary grade and feedback on your project. The OwlTest page has a pale yellow background and does **not** submit your project to Coursera. OwlTest is just meant to allow you to test your project automatically. Note that trying to debug your project using the tests in OwlTest can be very tedious since they are slow and give limited feedback. Instead, we *strongly* suggest that you first test your program using your own tests. Also, note that each OwlTest link is specific to a particular project. You need to come back to this page and click the link above to ensure that you are running the tests for this project.

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: Creating Line Plots of GDP Data

Provided Files

As part of its mission, the World Bank has maintained detailed records concerning the economic activity of countries in the world for over half a century. In particular, the World Banks has records of the Gross Domestic Product (GDP) by country from 1960-2015. The World Bank site includes lots of tools for visually analyzing this GDP data. This page at the World Bank data site allows users to plot the yearly GDP of a particular country in terms of current US dollars.

The raw economic data collected by the World Bank is freely available for download and analysis. In particular, the yearly data on GDP in current US dollars is available as a CSV file. We have downloaded a version of the CSV file which contains GDP data up until the end of 2015 and slightly updated this file to handle missing GDP data in a more consistent manner. Our version of the the CSV file is available here. **Please use this version for your project** since having everyone work from the same data set will make testing your code much easier.

Before starting the project, we suggest that you review this file in your favorite spread sheet program. Note that the first two columns corresponds to the "Country Name" and "Country Code" for each country in the file. Subsequent fields include GDP data (in current US dollars) for the years from 1960-2015 inclusive. Your task for this project is to write several functions that read the data in this CSV file into a dictionary keyed by country name and then plot the yearly GDP of a specified list of countries as an XY plot in Pygal.

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.

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. 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.

Working with the CSV file

As the format of the CSV file that stores the GDP data could change (or you could acquire data from somewhere else), the functions that operate directly on the data will all take a "gdpinfo" 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. If the years for which there is data within the file change, for instance, you can simply update the gdpinfo structure appropriately and all of your code will continue to work. Furthermore, if you have a CSV file that uses different field separators, you can tailor the gdpinfo structure appropriately to deal with that without needing to change any of your other code. The gdpinfo 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):

- "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. It looks as follows:

```
gdpinfo = {
    "gdpfile": "isp_gdp.csv",        # Name of the GDP CSV file
    "separator": ",",                # Separator character in CSV file
    "quote": '"',                    # Quote character in CSV file
    "min_year": 1960,                # Oldest year of GDP data in CSV file
    "max_year": 2015,                # Latest year of GDP data in CSV file
    "country_name": "Country Name",  # Country name field name
    "country_code": "Country Code"   # Country code field name
}
```

You will use this same gdpinfo structure for all of the projects throughout this course.

Problem 1: Read a CSV file as a nested dictionary

First, you will write a function called **`read_csv_as_nested_dict`** that takes the name of a CSV file and returns the data within the file as a dictionary of dictionaries. Each key-value pair in the outer dictionary corresponds to a row in the CSV file. The keys in that dictionary are the values of a header column in the table. The function takes the name of that header column as the input

keyfield. If a key appears multiple times in column corresponding to **keyfield**, the last row containing the key is used to create the dictionary used as the corresponding value.The inner dictionaries within the outer dictionary map the field names to the column values for that row. As CSV files can use different separator characters and quote characters, this function takes those characters as input and uses them to properly parse the CSV file. Note that the key-value pair for **keyfield** should be in the inner dictionaries, even though its value is used as the key in the outer dictionary.

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

```
def read_csv_as_nested_dict(filename, keyfield, separator, quote):
    """
    Inputs:
      filename  - Name of CSV file
      keyfield  - Field to use as key for rows
      separator - Character that separates fields
      quote     - Character used to optionally quote fields

    Output:
      Returns a dictionary of dictionaries where the outer dictionary
      maps the value in the key_field to the corresponding row in the
      CSV file.  The inner dictionaries map the field names to the
      field values for that row.
    """
```

You need to write the code that implements this function.

Hint:

1. You should already have code that implements this function available from the projects for the previous course, feel free to use it!
2. If you do not already have a working implementation, be sure to read the documentation on the **csv** module. In particular, you should read about how **csv.DictReader** works.

Problem 2: Create a list of GDP data suitable for XY plotting

Next, you will write a function called **build_plot_values** that takes a **gdpinfo**, a gdpinfo dictionary describing the CSV file that contains the GDP data, and **gdpdata**, a dictionary that contains a single country's GDP data mapped by year and returns a list of tuples suitable for XY plotting. The **gdpdata** dictionary maps years (which are strings) to GDP data for that year (also stored as strings). The tuples in the returned plot values list should be of the form **(year, gdp)** where **year** is an integer between the minimum and maximum years (inclusive) specified in **gdpinfo** and **gdp** is a float corresponding to the GDP for the given year.

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

```
def build_plot_values(gdpinfo, gdpdata):
    """
    Inputs:
      gdpinfo - GDP data information dictionary
      gdpdata - A single country's GDP stored in a dictionary whose
                keys are strings indicating a year and whose values
                are strings indicating the country's corresponding GDP
                for that year.

    Output:
      Returns a list of tuples of the form (year, GDP) for the years
      between "min_year" and "max_year", inclusive, from gdpinfo that
      exist in gdpdata.  The year will be an integer and the GDP will
      be a float.
    """
```

You need to write the code that implements this function.

Hints:

1. For some countries, World Bank GDP data is not available for the entire range of years. If there is no GDP for a particular year, that column would have been empty in the CSV file. This will become an empty string in the **`gdpdata`** dictionary. Such years should be omitted from the returned XY plot list.
2. Remember that everything in the **`gdpdata`** dictionary is stored as strings, but the returned XY plot list needs to contain integer years and floating point GDP data.

Problem 3: Create a dictionary mapping countries to XY plot lists

Third, you will write a function called **`build_plot_dict`** that takes **`gdpinfo`**, a gdpinfo dictionary describing the CSV file that contains the GDP data, and **`country_list`**, a list of strings which are country names, and returns a dictionary that maps all of the country names within **`country_list`** to plot value lists of GDP data for that country.

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

```
def build_plot_dict(gdpinfo, country_list):
    """
    Inputs:
      gdpinfo      - GDP data information dictionary
      country_list - List of strings that are country names

    Output:
      Returns a dictionary whose keys are the country names in
      country_list and whose values are lists of XY plot values
      computed from the CSV file described by gdpinfo.

      Countries from country_list that do not appear in the
      CSV file should still be in the output dictionary, but
      with an empty XY plot value list.
    """
```

You need to write the code that implements this function.

Hints:

1. Make sure that you utilize the functions that you have already written.
2. You need to think carefully about how you want to read the CSV file so that it is in a form that can be used by your other functions. In particular, pay attention to what you want to use as the key field.
3. If a country in **`country_list`** does not appear in the World Bank GDP data, this country should still appear in the returned dictionary, but it should map to an empty XY plot list

Problem 4: Create an SVG image of XY plots for yearly GDP for a specified list of countries

Finally, you will write a function called **`render_xy_plot`** that utilizes the first three functions you wrote to create an XY plot of the World Bank GDP data for a specified list of countries. This function will take **`gdpinfo`**, a gdpinfo dictionary describing the CSV file that contains the GDP data, **`country_list`**, a list of strings that name the countries that should be in the output plot, and **`plot_file`**, a string that names the output file the function should create. The output plot should be an SVG image of the resulting XY plot. To create create this plot, you should review Pygal's documentation on XY plots and outputting the results of the plot as an image file.

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

```
def render_xy_plot(gdpinfo, country_list, plot_file):
    """
    Inputs:
      gdpinfo      - GDP data information dictionary
      country_list - List of strings that are country names
      plot_file    - String that is the output plot file name

    Output:
      Returns None.

    Action:
      Creates an SVG image of an XY plot for the GDP data
      specified by info for the countries in country_list.
      The image will be stored in a file named by plot_file.
    """
```

You need to write the code that implements this function.

Hints:

1. Make sure that you have successfully installed Pygal before trying to test this function. (Also remember to **`import pygal`**.)
2. Use the Pygal render_to_file() output method to save the resulting XY plot as an SVG image. While testing, you may also wish to experiment with the render_in_browser() output method. Note that this method requires the **`lxml`** package to be installed.
3. The precise labels on your plots are up to you. As a general rule of thumb, your plot should have an informative title as well as informative labels for both the horizontal and vertical axes. See Pygal's documentation on how to add such labels to your plot.

VERY IMPORTANT FINAL NOTE: OwlTest does not include tests for the final function **`render_xy_plot`** 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 first three required functions. You are welcome to self-assess the correctness of your version of the final function **`render_xy_plot`** based on the sample output images provided below (note that the function **`test_render_xy_plot`** in the template will call your **`render_xy_plot`** function appropriately to create these images):

- **`country_list = []`** produces the XY plot isp_gdp_xy_none.svg.
- **`country_list = ["China"]`** produces the XY plot isp_gdp_xy_china.svg.
- **`country_list = ["United Kingdom", "United States"]`** produces the XY plot isp_gdp_xy_uk+usa.svg.

Mark as completed