World Bank API Project

Background on API

API's (Application Programming Interfaces) are interfaces provided to allow two applications to communicate with each other. As a developer, you can use API's provided by other companies/websites to access data from those companies/websites.

Some examples:

https://www.zillow.com/howto/api/APIOverview.htm

https://www.yelp.com/developers/documentation/v3/get_started

https://polygon.io/

Details about World Bank API

In this project, we will use the World Bank API to import world bank data into Python. The World Bank API provides a way for developers to access World Bank data.

The link below provides a bunch of links that document how to use the API:

https://datahelpdesk.worldbank.org/knowledgebase/topics/125589-developer-information

The link below documents how to set up your URL strings to make calls into the API:

https://datahelpdesk.worldbank.org/knowledgebase/articles/898581-api-basic-call-structures

The above links are provided for your reference, but if you follow my instructions below, you should be able to complete the project without reading the above web pages.

The World Bank data has a variety of indicators across many countries and many years including economic indicators and much more. You can see all of the indicators here: https://data.worldbank.org/indicator?tab=all. Each indicator has a string code. If you click on an indicator at the above website, and then click Details, you can see the code under "ID". The code is also visible in the URL in your web browser after clicking on the indicator. For example, if you click on "Access to electricity, rural", you will be taken to the following web address:

https://data.worldbank.org/indicator/EG.ELC.ACCS.RU.ZS?view=chart

The code for the "access to electricity, rural" indicator is EG.ELC.ACCS.RU.ZS.

The World Bank API allows for downloading the data for all indicators in either XML or JSON format. For this project, we will use the XML version and use BeautifulSoup to import this data into Python data frames.

The Project

You should write a function called **worldbankimport** in a file called **worldbankimport.py** that takes two arguments, an indicator code (of type string) and a year (of type int) and imports the data for that code/year (for all countries) into a python data frame. The function should return the data frame.

As an example, consider the per-capita GDP indicator, which has the code: NY.GDP.PCAP.CD. Below I have examples of how to set up your URL string to download data for this indicator.

If we wanted to get per-capita GDP for all countries in all years (in XML format), use the following URL:

https://api.worldbank.org/v2/country/all/indicator/NY.GDP.PCAP.CD

For this project, we want to get data one year at a time (based on the year argument of your function). So, to get per-capita GDP for all countries in 2010 (in XML format), use the following URL:

https://api.worldbank.org/v2/country/all/indicator/NY.GDP.PCAP.CD?date=2010

Both of the examples above get only the first page of data. The first tag in the XML file, if you use either of the URL's above, has an attribute called "pages" that indicates how many pages of data there are. To get the second page, use the following URL:

https://api.worldbank.org/v2/country/all/indicator/NY.GDP.PCAP.CD?date=2010&page=2

Note that the URL in all three examples above start out the same. In this project, your URL will always begin with the string: https://api.worldbank.org/v2/country/all/indicator/

Then you will add the correct indicator code and year to get the data, following the examples above for the correct syntax. And make sure to download all pages of data!

Tips:

1. If you click on any of the links for the three GDP examples above, note that the very first tag has an attribute called "pages" that tells you how many pages of data there are. You will need to use this attribute to know how many pages to import, to make sure you get all of the data.
2. If you inspect the hierarchy of the XML files in BeautifulSoup, you will notice that there is a tag for the entire page, and that tag has one child for each row of data, and each row tag has one child for each column (columns are indicator, country id, etc.). But, you will also see that every other child is just a string "\n" instead of a Tag. You want to ignore those "\n" children. Instead of using tag.contents (which is a list of all children including "\n"), you can use tag.find_all(True, recursive=False). This returns a list of all direct children that are Tags (ignoring direct children that are strings like "\n").
3. The code for my XML example in class is a good place to start. You need to modify it to get the right URL string (following example above), to loop over multiple pages (tip 1 above), and to deal with the "\n" (tip 2).