For our second practice project, our task is to compute position of country centers on the USA map that we plotted in the previous week. In subsequent weeks, we will plot the cancer-risk data for each county on this map using the county centers that we compute this week. To get started, we suggest that you examine the SVG version of the USA map in a text editor such as Notepad++.

The key feature to note concerning this SVG image is that its file consists of plain text organized in the XML format. XML consist of a logical collection of *elements*, each of whom have associated *attributes*. For this particular file, we observe that the boundary for each county in the map corresponds to a `<path ... />` element in the XML. Moreover, each path element includes two attributes: the `id` attribute (the path ID) stores the FIPS code for the county while the `d` attribute (the path data) stores a string that encodes the coordinates for the boundary of the county.

Practice Project: Extracting Data from an SVG File

Our task for this week's practice project has three steps:

- Parse the XML in the SVG file using an existing Python module and extract the `d` and `id` attributes for each county.
- Extract the county's boundary coordinates (as a list of map coordinates) from the path data for each county,
- Compute the center of each county using our provided code and output a CSV file with FIPS codes and county centers.

Parse the XML in the given SVG file

For this first step, we have several options. One option is to read in the file and extract the required attributes using basic string processing. At this point in the specialization, you should probably be able to succeed with this strategy. However, as a Python scripter, you should always explore the possibility that Python provides existing functionality to make this task easier.

Searching with the keywords `"python 3 svg parse"` yields an extremely helpful Stack Overflow post on parsing XML in Python. The essence of this post is that Python 3 support a built-in `xml` module that provides very nice functionality for parsing an XML file. In particular, the module xml.dom.minidom implements several methods that are very useful for this task.

For the first part of the practice project, your task is to write a function `get_county_attributes(svg_file_name)` that takes the name of the USA SVG file and returns a list of tuples where each tuple is a pair of strings corresponding to the `id` and `d` attributes of a county. Test this function using the test code in the provided template.

Extract county boundary coordinates from the path data

Having completed the first part of this project, our next task is to process the path data (i.e. the `d` attribute) for each path element and extract a list of coordinates that correspond to the boundary of the county. While format of the path data is fairly complex, our approach will be to treat the path data as a sequence of pairs of floats separated by non-numeric characters corresponding to path commands and extract these pairs of floats while ignoring the commands in the path data.

For the second part of this practice project, your task is to write a function `get_boundary_coordinates(boundary_data)` that takes path data (as a string) and return a list of boundary coordinates where each coordinate is itself a pair of floats. Your code should ignore the various commands (`&#x27;L&#x27;`, `&#x27;M&#x27;`, `&#x27;z&#x27;`) in the path data and simply extract the numerical coordinates in the path data using basic string processing.

Once you have implemented this function, test this function using the code in the provided template. Note that this test code calls a function `compute_county_center(boundary_coordinates)` that we have provided which takes a list of coordinates for a closed polygon and returns the coordinates for the center of this polygon. Note that, in practice, the coordinates encoded in the path data do not always form a closed polygon. However, this provided code will usually still compute an excellent approximation to the county's center.

Compute county boundaries and output the results to a CSV file

With these two functions in hand, your final task is to write a script `process_county_attributes(svg_file_name, csv_file_name)` that parses the XML in the named SVG file, computes county centers, and writes the results to the named CSV file. The output CSV file should be comma-separated and have one row for each county. The three rows of the file should contain county FIPS code, horizontal coordinate for county center, and vertical coordinate for county center.

Now, run this script on the SVG version of the USA map. We have not provided you with any test code for this script. However, we suggest that, minimally, you examine the resulting CSV file using a text editor to ensure that it has the correct format. In particular, the file should have exactly 3143 rows and 3 columns. Once you are done, you are welcome to examine our solution code.

Mark as completed