```
#IMPORTANT: This is the Python library portion.
#It is necessary for the Javascript portion to work properly.



#Shapefile reader built off PySHP library which was not made by me.
import shapefile
import json

class Coordinates(dict):
    #Most of the __init__ function was written by me, but prior to the start of the performance task
for another project. It is not used in any written responses.
    def __init__(self,file,recordId,res=5,isl=1,countries=[],zoom=1,translate=
[0,0],height=180,width=360):

    #Open shapefile with PySHP
        self.shp = shapefile.Reader(file)
        shapes = self.shp.shapes()

        #Define some variables
        svg = []
        svgPoints = []
        recordList = []

        #Counter variable to track progress
        i = 0

        #Loop through each country or territory
        for country in shapes:
            i = i + 1

            #Restrict parsing to specific countries
            if((self.shp.shapeRecord(i-1).record[recordId] in countries) or countries==[]):

                #Track progress
                print(i)

                #Reset/define some variables
                main = []
                rounded = []

                #Loop through each point
                for point in country.points:

                    #Find lat and long
                    data = ','.join(str(x) for x in point)
                    long = float(data.split(',')[0])
                    lat = float(data.split(',')[1])

                    #Convert to positive numbers
                    long = long + 180
                    lat = lat + 90

                    #Convert original unrounded data to string
                    data = str(long) + ',' + str(lat)
                    data = data + ' '

                    #Check if point appears twice. If so, it is a complete polygon
                    if data in main:
                        main.append(data)

                        #Round to reduce filesize
                        pointRounded = [str(round(long*res)/res),str(round(lat*res)/res)]
```

```
                        #List of rounded points
                        rounded.append(pointRounded)

                        #Exclude small islands
                        if(len(rounded) > isl):

                            #Add shape to list
                            svgJson = {'Points':rounded,'Country':self.shp.shapeRecord(i-
1).record[recordId].replace(' ','')}
                            svg.append(svgJson)

                        #Reset lists
                        main = []
                        rounded = []
                    else:

                        #Round data as before and save the point
                        main.append(data)
                        roundedData = [str(round(long*res)/res),str(round(lat*res)/res)]
                        if not(roundedData in rounded):
                            rounded.append(roundedData)
            recordList.append(self.shp.shapeRecord(i-1).record[recordId].replace(' ',''))

        #Create object from the points
        self['data'] = svg
        self['metadata'] = {'scale':zoom,'translate':translate,'height':height,'width':width}
        self['recordList'] = recordList

#Builds the MapData class
class MapData(dict):
    def __init__(self,data,tlist='territory_list'):
        if(type(data) is dict):
            self['data'] = data
            try:
                #Finds which list is the territory list and marks it
                self['territory_list'] = self['data'][tlist]
            except:
                raise IndexError('Territory list not found in data.')
        else:
            raise TypeError('Data must be a dictionary.')

#Builds the infobox class
class Infobox(dict):
    def __init__(self):
        #Sets style information
        self['style'] = {}
        self['style']['border'] = {}
        self['style']['border']['stroke-width'] = 3
        self['style']['border']['color'] = {}
        self['style']['border']['color']['r'] = 0
        self['style']['border']['color']['g'] = 0
        self['style']['border']['color']['b'] = 0
        self['style']['color'] = {}
        self['style']['color']['r'] = 255
        self['style']['color']['g'] = 255
        self['style']['color']['b'] = 255
        self['style']['text'] = []
        self['text'] = []


    #IMPORTANT: Text is based off of text containers that contain some text all formatted the same
way. There can be multiple text containers in a line. The text containers can contain text or data.

    def addText(self,text,textType='text',line=None,textId=None,font='Calibri',size=15,r=0,g=0,b=0):
```

```
        #Create text
        textDict = {}
        textDict['type'] = textType

        #Assign unique ID
        if textId == None:

            #Default ID is just a number, but a string
            textDict['id'] = str(len(self['text']))
        else:

            #Custom ID
            textDict['id'] = textId

        #Decide which line text belongs on
        if line == None:
            if len(self['text']) == 0:

                #Text starts on first line
                textDict['line'] = 0
            else:

                #Text continues on same line as before
                textDict['line'] = self['text'][-1]['line']
        else:

            #Decide line manually
            textDict['line'] = line

        textDict['text'] = text

        #Create styles
        styleDict = {}
        styleDict['font'] = font
        styleDict['size'] = size
        styleDict['r'] = r
        styleDict['g'] = g
        styleDict['b'] = b

        #Add to object
        self['style']['text'].append(styleDict)
        self['text'].append(textDict)

    def removeText(self,textId):
        if type(textId) is int:
            del self['style']['text'][textId]
            del self['text'][textId]
        else:
            for text in self['text']:
                if(text['id'] == textId):
                    del self['style']['text'][self['text'].index(text)]
                    del self['text'][self['text'].index(text)]

class SVGMap(dict):
    def __init__(self,data=None,coords=None):
        self['map'] = coords
        self['choropleth'] = {}
        self['choropleth']['source'] = None

        self['choropleth']['colors'] = {}
        self['choropleth']['colors']['r'] = [0,255]
        self['choropleth']['colors']['g'] = [0,255]
        self['choropleth']['colors']['b'] = [0,255]

        self['layers'] = []
```

```
    self['infobox'] = {}

    #Supports two inputs: Hovering and clicking
    self['inputs'] = {}
    self['inputs']['onHover'] = ''
    self['inputs']['onMouseOut'] = ''
    self['inputs']['onClick'] = ''
    self['data'] = data

    #Code returned by returnCode is the output of the library portion.

def returnCode(self):
    #Saves the Javascript code and defines the variable data to be the dataset
    #as seen in the Javascript portion.
    return 'var data = ' + json.dumps(self) + ';'
```