Exercise 14: Writing Data to a Feature Class with Insert Cursor

As you parse out geographic information from "raw" sources such as text files, you may want to convert it to a format that is native to your GIS. In this exercise you will learn how how to write vector geometries to ArcGIS feature classes. We'll read through the same GPS-produced text file from the previous exercise, but this time we'll add the extra step of writing each coordinate to a polyline shapefile.

In this exercise you will be writing point geometries by first creating a Point object using Arcpy.CreateObject(). You will then you use an insert cursor to assign it to the geometry field of the feature class (called "shape" for shapefiles). The figure below shows an example of how this is done.

```
# Create point
inPoint = arcpy.CreateObject("Point")
inPoint.X = -121.34
inPoint.Y = 47.1

# newRow originates from an insert cursor
newRow.shape = inPoint
```

For polylines and polygons, you create multiple Point objects which you add to an Array object. The code below could be used to create either a polyline or polygon, depending on the geometry type for the feature class; however, with polygons it's a good practice to make the end vertex the same as the start vertex if possible.

```
vertexArray = arcpy.CreateObject("Array")

# Create some points
inPoint1 = arcpy.CreateObject("Point")
inPoint1.X = -121.34
inPoint1.Y = 47.1

inPoint2 = arcpy.CreateObject("Point")
inPoint2.X = -121.29
inPoint2.Y = 47.32

inPoint3 = arcpy.CreateObject("Point")
inPoint3.X = -121.31
inPoint3.Y = 47.02

vertexArray.add(inPoint1)
vertexArray.add(inPoint2)
vertexArray.add(inPoint3)

newRow.shape = vertexArray
```

Of course, you usually won't create points manually in your code like this with hard-coded coordinates. It's more likely that you'll parse out the coordinates from a file or capture them from some external source, such as a series of mouse clicks on the screen.

Creating a polyline from a GPS track

Here's how you could parse out coordinates from a GPS-created text file like the one in the previous section of the lesson. This code reads all the points captured by the GPS and adds them to one long polyline. The polyline is then written to an empty, pre-existing polyline shapefile with a geographic coordinate system named tracklines.shp. If you didn't have a shapefile already on disk, you could use the Create Feature Class tool to create one with your script.

```
# Reads a GPS-produced text file and writes the lat and long values
#  to an already-created polyline shapefile
import arcpy

# Hard-coded variables for GPS track text file and feature class
gpsTrack = open("C:\\Data\\GPS\\gps_track.txt", "r")
polylineFC = "C:\\Data\\GPS\\tracklines.shp"

# Figure out position of lat and long in the header
headerLine = gpsTrack.readline()
valueList = headerLine.split(",")

latValueIndex = valueList.index("lat")
lonValueIndex = valueList.index("long")

# Create an array to store the points for the polyline
vertexArray = arcpy.CreateObject("Array")
# Read each line in the file
for line in gpsTrack.readlines():
    segmentedLine = line.split(",")
    # Get the lat/lon values of the current GPS reading
    latValue = segmentedLine[latValueIndex]
    lonValue = segmentedLine[lonValueIndex]
    # Create a point and add it to the array
    vertex = arcpy.CreateObject("Point")
    vertex.X = lonValue
    vertex.Y = latValue
    vertexArray.add(vertex)
# Write the array (which now makes a polyline) to the feature class
cursor = arcpy.InsertCursor(polylineFC)
feature = cursor.newRow()
feature.shape = vertexArray
cursor.insertRow(feature)
del cursor
```

The above script starts out the same as the one in the previous section of the lesson. First, it parses the header line of the file to determine the position of the latitude and longitude coordinates in each reading. But then, notice that an array is created to hold the points for the polyline:

```
vertexArray = arcpy.CreateObject("Array")
```

After that, a loop is initiated that reads each line and creates a point object from the latitude and longitude values. At the end of the loop, the point is added to the array.

```
for line in gpsTrack.readlines():
    segmentedLine = line.split(",")

    # Get the lat/lon values of the current GPS reading
    latValue = segmentedLine[latValueIndex]
    lonValue = segmentedLine[lonValueIndex]

    # Create a point and add it to the array
    vertex = arcpy.CreateObject("Point")
    vertex.X = lonValue
    vertex.Y = latValue
    vertexArray.add(vertex)
```

Once all the lines have been read, the loop exits and an insert cursor is created. The cursor is used to create a new row, or polyline, but this polyline has no geometry information yet. The vertexArray object is assigned to the shape field, thereby giving the row some geometry.

```
cursor = arcpy.InsertCursor(polylineFC)
feature = cursor.newRow()
feature.shape = vertexArray
cursor.insertRow(feature)

del cursor
```

Remember that the cursor places a lock on your dataset, so this script doesn't create the cursor until absolutely necessary (in other words, after the loop). After the row is inserted, the cursor is deleted to remove the lock.

Extending the example for multiple polylines

Suppose your GPS allows you to mark the start and stop of different tracks. How would you handle this in the code? Open the modified text file for the GPS track (gps_track_multiple.txt) and examine the data

Notice that in the GPS text file, there is an entry **new_seg**:

*type,ident,lat,long,y_proj,x_proj,**new_seg**,display,color,altitude,depth,temp,time,model,filename,ltime*

new_seg is a boolean property that determines whether the reading begins a new track. If new_seg = true, you need to write the existing polyline to the shapefile and start creating a new one. Take a close look at this code example and notice how it differs from the previous one in order to handle multiple polylines:

```
# Reads a GPS-produced text file and writes the lat and long values
#  to an already-created polyline shapefile. Handles multiple polylines.

import arcpy

# Hard-coded variables for GPS track text file and feature class
gpsTrack = open("C:\\Users\\Me\\Desktop\\Getting to know Python\\Data\\gps_track_multiple.txt", "r")
polylineFC = "C:\\Users\\Me\\Desktop\\Getting to know Python\\Data\\tracklines_multiple.shp"
```

```
# Functions to add a vertex and add a completed polyline to the shapefile
def addVertex(lat, lon, array):
  vertex = arcpy.CreateObject("Point")
  vertex.X = lon
  vertex.Y = lat
  array.add(vertex)

def addPolyline(cursor, array):
  feature = cursor.newRow()
  feature.shape = array
  cursor.insertRow(feature)
  array.removeAll()

# Figure out position of lat and long in the header
headerLine = gpsTrack.readline()
valueList = headerLine.split(",")

latValueIndex = valueList.index("lat")
lonValueIndex = valueList.index("long")
newTrackIndex = valueList.index("new_seg")

# Read lines in the file and append to coordinate list
cursor = arcpy.InsertCursor(polylineFC)
vertexArray = arcpy.CreateObject("Array")

# Read each line and split it
for line in gpsTrack.readlines():
  segmentedLine = line.split(",")
  isNew = segmentedLine[newTrackIndex].upper()

  # If starting a new line, write the completed
  #  line to the feature class
  if isNew == "TRUE":

     # This check is needed to handle the first GPS entry
     if vertexArray.count > 0:
        addPolyline(cursor, vertexArray)

  # Get the lat/lon values of the current GPS reading
  latValue = segmentedLine[latValueIndex]
  lonValue = segmentedLine[lonValueIndex]

  # Call the function we defined earlier to add a vertex
  addVertex(latValue, lonValue, vertexArray)

# Add the final polyline to the shapefile
addPolyline(cursor, vertexArray)

del cursor
```

The first thing you should notice is that some focused pieces of functionality have been put inside functions. The addVertex function adds a vertex to an array. It helps make the code shorter and more readable. The addPolyline function adds a completed array to the shapefile as a polyline. It's included as a function to avoid repeating code.

Here's a look at the addVertex function:

```
def addVertex(lat, lon, array):
    vertex = arcpy.CreateObject("Point")
    vertex.X = lon
    vertex.Y = lat
    array.add(vertex)
```

The addVertex function helps shorten the code inside the loop (which has grown longer now that we have to check for a new track each time). If you had to write some more advanced logic that added vertices to the array in several different branches of your code, a function would be even more useful.

Notice it's okay to use arcpy in the above function, since it is going inside the body of a script that imports arcpy. However, you want to avoid using variables in the function that are not defined within the function or passed in as parameters.

Here's a look at the addPolyline function:

```
def addPolyline(cursor, array):
    feature = cursor.newRow()
    feature.shape = array
    cursor.insertRow(feature)
    array.removeAll()
```

The addPolyline function above takes in an insert cursor and an array as parameters. It creates a new row in the shapefile and applies the array to the shape field. Then it does some cleanup by clearing out the array. Notice that the addPolyline function is called twice in the script: once within the loop, which we would expect, and once at the end to make sure the final polyline is added to the shapefile. This is where writing a function cuts down on repeated code.

As you read each line of the text file, how do you determine whether it begins a new track? First of all, notice that we've added one more value to look for in this script:

```
newTrackIndex = valueList.index("new_seg")
```

The variable newTrackIndex shows us which position in the line is held by the boolean new_seg property that tells us whether a new polyline is beginning. If you have sharp eyes, you'll notice we check for this later in the code:

```
segmentedLine = line.split(",")
isNew = segmentedLine[newTrackIndex].upper()

# If starting a new line, write the completed
#  line to the feature class
if isNew == "TRUE":
```

In the above code, the upper() method converts the string into all upper-case, so we don't have to worry about whether the line says "true", "True", or "TRUE". But there's another situation we have to handle: What about the first line of the file? This line should read "true", but we can't add the existing polyline to the file at that time, because there isn't one yet. Notice that a second check is performed to make sure there are more than zero points in the array before the array is written to the shapefile:

```
# Need this > 0 check to handle the first track
if vertexArray.count > 0:
    addPolyline(cursor, vertexArray)
```

The above code checks to make sure there's at least one point in the array, then it calls the addPolyline function, passing in the cursor and the array.

Here's another question to consider: How did we know that the Array object has a count property that tells us how many items are in it? This comes from the http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v0000005r000000.htm In this section of the help there are topics describing each class in arcpy, and you'll come here often if you work with ArcGIS geometries in Python.

In the above-linked Array topic, find the Properties table in this topic and notice that Array has a read-only count property. If we were working with a Python list, we could use len(vertexArray), but in our case vertexArray is an Array object that is native to the ArcGIS geoprocessing programming model. This means it is a specialized object designed by Esri, and you can only learn its methods and properties by examining the documentation. Bookmark these pages!