Exercise: Getting data from files

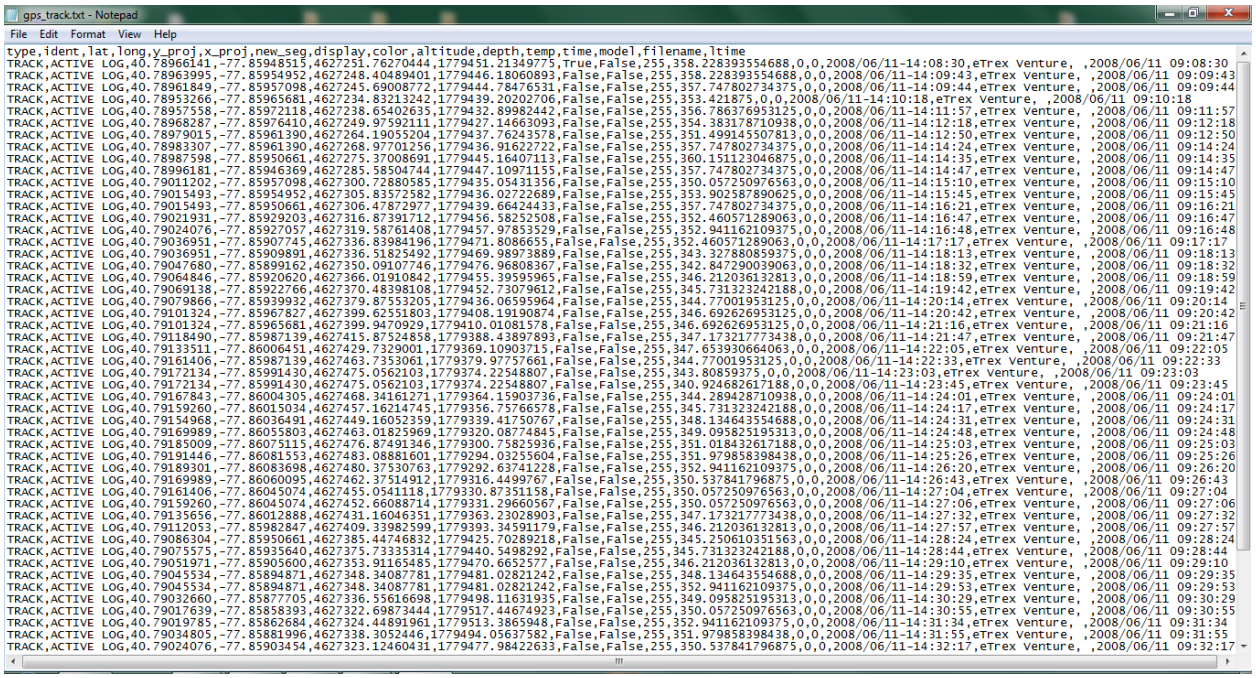
One of the best ways to increase your effectiveness as a GIS programmer is to learn how to manipulate text-based information. GIS data is often collected and shared in text-based formats such as an Excel spreadsheet in CSV (comma-separated value) format, a list of coordinates in a text file, or an XML response received through a Web service.

When faced with these files, you should first understand if your GIS software already comes with a tool or script that can read or convert the data to a format it can use. If no tool or script exists, you'll need to do some programmatic work to read the file and separate out the pieces of text that you really need. This is called parsing the text. When you parse, you cycle through lines of text, treating them as strings, and pull out the useful information from those strings. In an XML file, for example, you may know that the information you want falls inside a particular tag, such as <AvgTemp>46</AvgTemp>. One approach to getting the value 46 might be to search for the line containing the substring "AvgTemp", then take all the characters that fall between the first > coming from the left of the string and the first < coming from the right.

In another case, you might know that the values you want fall after the second and third commas in a line of comma-separated values. You can split up the line based on comma locations and put all the segments of the string in a Python list. You can then take the third and fourth items in the list to get the values you want. (Remember the third and fourth items would come after the second and third commas, respectively.) In both cases, the keys to effective parsing are to know how to read lines in a file and know your string manipulation methods. It's helpful to know how to read a string, search for values in a string, split up a string based on some delimeter, and extract particular segments of a string.

Sometimes you can import helper modules, or libraries, into your code to make it easier to parse certain types of text. In this exercise, you will read a text file collected from a GPS unit. The lines in the file represent readings taken from the GPS unit as the user traveled along a path. The method used in this exercise is one of many that can be used to parse out the coordinates from each reading.

The file for this example is called gps_track.txt. The figure below shows a snapshot of what the contents of this file looks like. Open the file and examine its contents.



Notice that the file starts with a header line, explaining the meaning of the values contained in the readings from the GPS unit. Each subsequent line contains one reading. The goal in this exercise is to create a Python list containing the X,Y coordinates from each reading. Specifically, the script should be able to read the above file and print a text string like the one shown below.

[['-77.85948515', '40.78966141'], ['-77.85954952', '40.78963995'], ['-77.85957098', '40.78961849'], etc.]

Step 1: Plan approach for parsing the GPS track

Before you start parsing a file, it's helpful to outline what you're going to do and break up the task into manageable chunks. Here's some pseudocode for the approach we'll take in this example:

1. Open the file.
2. Read the header line.
3. Loop through the header line to find the index positions of the "lat" and "long" values.
4. Read the rest of the lines.
5. Split each line into a list of values, using the comma as a delimiter.
6. Find the values in the list that correspond to the lat and long coordinates and write them to a new list.

Step 2: Opening the file

The first thing the script needs to do is open the file. Python contains a built-in open() method for doing this. The parameters for this method are the path to the file and the mode in which you want to open the file (read, write, etc.). In this exercise you will only need to read the contents of the file and print them out to the user. "r" stands for read-only mode, “w” for write mode, etc.

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")
```

Step 3: Reading the header line
Opening the file with the open() method gets you a file object (called gpsTrack in our case). You can read the first line by calling the file.readline() method, like this:

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")

# Figure out position of lat and long in the header
headerLine = gpsTrack.readline()
```

This returns the string:

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

Step 3: Looping through the header line to find the index positions of the the "lat" and "long" values
You need to search through this string and find the position of "lat" and "long". If you start counting comma-separated values in this string beginning from zero, it's easy to see that "lat" is at index position 2 and "long" is at index position 3. However, it's a good practice not to hard-code numbers like 2 and 3 into your script. Hard-coded numbers other than 0 or 1 are sometimes derided as *magic numbers*, suggesting that if you're not the programmer, you might have to use magic to know where the numbers came from!

Avoiding magic numbers gives you greater flexibility. If you wanted to re-use this script with a file in which "lat" and "long" were in different positions, you wouldn't have to modify your code. Even if "lat" and "long" went by some other name, it would be easier to find and change a string in your script instead of finding and changing a "magic number".

So how can you programmatically determine that "lat" is at index 2 and "long" is at index 3? Below is one way that uses the string.split() method. This method puts each "item" in the line into a list. The parameter you pass to the split method determines the delimiter, or the character that determines a new list item. In our case, it's the comma:

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")

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

The above method call returns:

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

The key is that now you can cycle through this list and discover the position of "lat" and "long". To do this, you could write a loop that searched through the list for "lat" and "long", but a quicker way is to use the helper method index() that gets you the index position of any item in the list:

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")

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

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

After running the above lines, latValueIndex is equal to 2 and lonValueIndex is equal to 3. With those variables set, you're now ready to start reading the rest of the lines in the file.

Step 4: Processing the rest of the lines in the file

When you have an open text file, you can always call file.readline() to go to the next line. In our case, we know we're going to use all the rest of the lines in the file, so it's more efficient to call file.readlines() to read them all at once. (This might not be efficient with an extremely long file.) The readlines() method returns a list of all the remaining lines in the file.

Now you can cycle through each GPS reading and split it up based on commas the same way you split up the header. You specifically need to pull out the values in index positions 2 and 3 of your list (represented by latValueIndex and lonValueIndex, respectively) and write those to a new list (coordList).

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")

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

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

# Read lines in the file and append to coordinate list
coordList = []

for line in gpsTrack.readlines():
    segmentedLine = line.split(",")
    coordList.append([segmentedLine[lonValueIndex], segmentedLine[latValueIndex]])
```

Notice that we first created the list and then placed values into it. “coordList” actually contains a bunch of small lists within a big list. Each small list is a coordinate pair representing the x (longitude) and y (latitude) location of one GPS reading. The list.append() method is used to add items to coordList. Notice again that you can append a list itself (representing the coordinate pair) using this method.

Step 5: Printing the coordinate values

```
# Reads a GPS-produced text file and writes the lat and long values
#  to a list of coordinates
gpsTrack = open("C:\\Users\\Me\\Desktop\\GIS Programming\\Training\\Data\\gps_track.txt", "r")

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

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

# Read lines in the file and append to coordinate list
coordList = []

for line in gpsTrack.readlines():
    segmentedLine = line.split(",")
    coordList.append([segmentedLine[lonValueIndex], segmentedLine[latValueIndex]])

print coordList
```

Step 6: Save and run script

Save the script as getGPScoords.py and run the script