Name: ______________________ Date: ______________________

Lab 4: User-Defined Functions

Chapters in Learning Scientific Programming with Python (LSPP):
Matplotlib, Chapter 7.1.1 – 7.1.4
numpy, Chapter 6.1 and 6.2
Functions, Chapter 2.7

This laboratory exercise will involve you writing your first Python functions. Make sure to comment your code and include the proper Python header info!

Part I: Your First Function: Volume of a Sphere

1. Open a new Python file called `earthLayers.py`. In this file, you should write a *function* that accepts one argument, radius, and returns the volume of a sphere with the user-specified radius. The formula for the volume of a sphere is $V = \frac{4}{3}\pi r^2$, where r is the radius of the sphere. This should be a very simple function to write. For example, other than comments and the return statement, my function has only one line of code.

2. In the same file, after your header info, copy in the following lines of code:

```
%thicknesses of Earth's layers in km
crust=35
mantle=2850
core=3486
```

   Once you have the appropriate radii calculated and stored for each layer, you must call your `volSphere` function to calculate the volumes. Use the provided layer thicknesses to calculate the volumes of each layer of the Earth (3 in all), and Earth's total volume. Store the appropriate values in variables named something like `volCore`, `volMantle`, etc. . . . This script should be generalized, so that if you wanted to calculate the volumes of crust, mantle, and core on some other planet, all you would have to change are the values assigned to `crust`, `mantle`, and `core`.

3. Once you have calculated the layer volumes and Earth's total volume (in km^3), you can calculate what percent of Earth's total volume each layer occupies. Store these values in variables named something like `pctVolCrust`, `pctVolMantle`, etc. . . .

4. Your script should end by printing out the following information to the command window.

```
Volume of Core: x.xxe + 10 (units)
Volume of Mantle: x.xxe + 10 (units)
Volume of Crust: x.xxe + 10 (units)
Core: xx.x %
Mantle: xx.x %
Crust: xx.x %
```

Page 36 of LSPP discusses formatting output. Replace the xx.xx with your calculated values. The volumes should be printed in units of km^3 in *scientific notation* with *two decimal places*. The volume percent's must be labeled with a percent sign and should be printed in *floating point notation* with *one decimal place* of precision. You will have to use an *escape sequence* with the percent sign. See page 28 of LSPP. Make sure that your *values all line up* at the decimal places when printed to the screen (like shown above). This will make the output easier to read.

5. Just for fun, have your script make a pie chart (using the command `plt.pie`) of the total % volumes of each layer. Use the colormap called `summer`, and explode out each layer, so they are easy to see. Otherwise, your crust slice will be very small and hard to see. `plt.pie()` might automatically label the %'s of each layer (I haven't tried this), but add a text label for each slice (i.e. Crust, Mantle, Core) using the `plt.text()`. For details on these new commands (`plt.text`, `plt.pie`), read Chapter 7.1 of LSPP, read the documentation for each command (`plt.pie?`), and experiment with different options.

   `pie` and `text` are both relatively straightforward, so you shouldn't have too much trouble with them.

Part II: Automated GPS Data Processing

1. Download and extract the lab file provided on the course website. The files are the position time series of two continuous GPS sites in Iceland. See the map below for locations. Your task is to write a *function* called `getGPSstats` that will calculate several useful parameters from the data in the file and print them to the screen. The *script* will call the function that prints the parameters and then make plots of the north, east, and vertical positions over time.

2. The data in each filename.tenv3 file is consistently organized into columns. Relevant columns are below:

Col 1:	Station
Col 2:	Date (YYMMDD)
Col 9:	East Position (meters)
Col 11:	North Position (meters)
Col 13:	Vertical Position (meters)
Col 15:	East Position Error (meters)
Col 16:	North Position Error (meters)
Col 17:	Vertical Position Error (meters)

   Load all of this data into Python and store it in an array called `data_HOFN` for the HOFN.IGS08.tenv3 file and `data_REYK` for the RYEK.IGS08.tenv3 file. You do not need to do anything with the errors, so you can ignore these data. Note that this type of GPS data gives one data point per day, so each data point represents one day.

3. In your *function*, calculate the following parameters from the data. Keep in mind that velocity is a vector, so a negative east velocity implies a westward motion.

   - Station Name (can be grabbed from the first column of the file; see Chapter 2.3.4 LSPP for string processing tips)

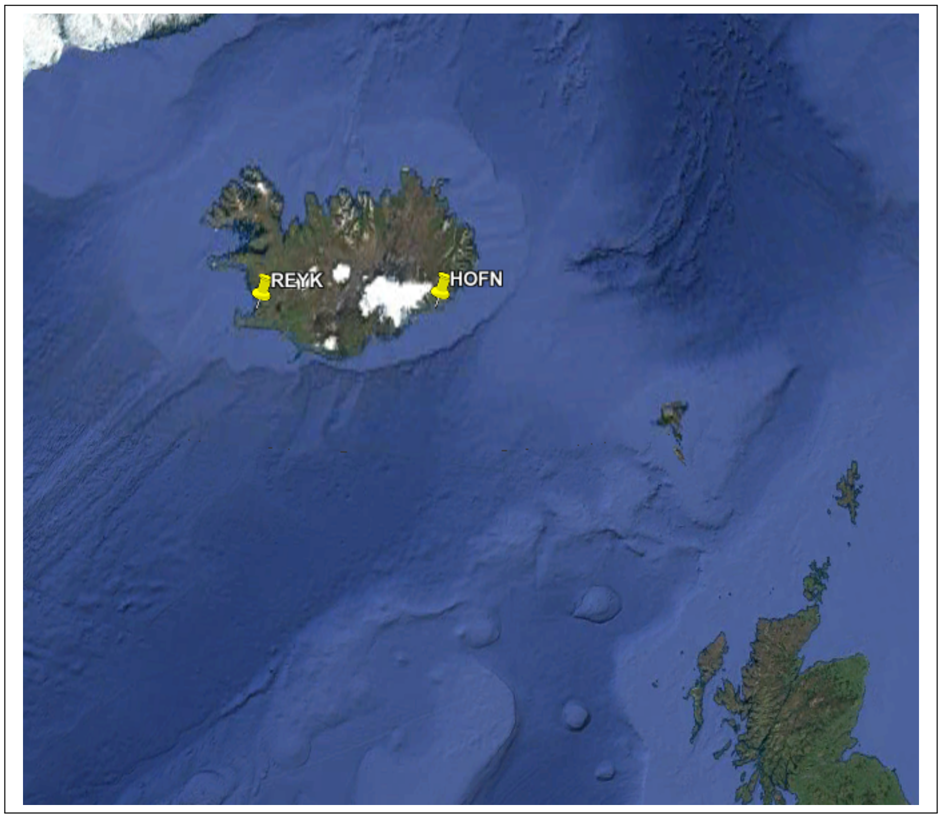

Figure 1: Location of the GPS stations in Iceland. Ask a geologist the significance of two locations. Why are they on opposite sides of Iceland?

- Total Time (decimal years)
- Number of Days in the Time Series
- Total East Displacement (i.e. final position — initial position, cm)
- Total North Displacement (cm)
- Total Vertical Displacement (cm)
- Average East Velocity (i.e. Total East Displacement/Total Time, cm/yr)
- Average North Velocity (cm/yr)
- Average Vertical Velocity (cm/yr)

(*NB: I have asked you to change the units of the data.*)

This information *should then be printed out to the screen* in an organized format with all values shown in *floating point format to only two decimal places*, except for site name and number of data points (days), which should be printed as a string and integer, respectively. For example:

```
Site name: xxxx
Time span: xx.xx (yrs)
Number of days with data: xxxx
Total north displacement: xx.xx (cm)
Total east displacement: xx.xx (cm)
```

Total vertical displacement: xx.xx (cm)
Avg north velocity: xx.xx (cm/yr)
Avg east velocity: xx.xx (cm/yr)
Avg vertical velocity: xx.xx (cm/yr)

4. Your *script* should also make three separate plots (arranged vertically — see `plt.subplot()`) in the *same figure window*. The top plot will be decimal time (yrs) for the x-axis and north position (cm) as the y-axis. The next two plots will be the same, but for east and vertical positions. Make sure to label each axis. Plot the HOFN position time series with a red line. Make the REYK data a blue line. Your plots might look quite squished in the default figure window, but if you use `plt.tight_layout()` at the end of the plotting calls and just before `plt.show()`, the figure everything should look fine.

5. Write a brief script called `allGPS.py` that calls your `getGPSstats.py` function two times (one for each GPS file).

Part III: What to Hand in?

Please turn in your scripts as scriptname_yourname.py. Please don't use spaces in your file name. Linux does not handle spaces well, so no spaces will make my job easier. Thanks.