ISSCENS
Lab 8

You can choose your language for this lab. If you will be working with models I recommend Fortran; otherwise work on the Python project.

Project 0 (For Fortran Programmers)

If you have a crufty old code from your advisor, improve it.

Project 1A (For Fortran Programmers).

Download the file vabirds.csv from Resources/Data.

1. Create a derived type bird_data in a module bird_dat. This is still a type; we do not need a class for this exercise. It should be in its own module, however.

Attributes:
One member of the type will be the species (a character variable) and the other member will be the list of observations. This should take the form of an allocatable array.

Methods:
A. Write a subroutine to act as the constructor for the type. Its parameters will be the species name and the array of input data. Allocate the type array based on the size of the input array.

B. Write a stats method that takes only an instance of the type and returns the mean and standard deviation of the observations for that instance.

C. Write a minmax method that takes an instance of the type and the array of years and returns the maximum observed, the minimum observed, and the years for maximum and minimum. You may use the maxval, minval, maxloc, and minloc intrinsics.

2. Write a main program that uses your module and also uses the sorters module I have posted to Resources/Fortran.

Remember to write an explicit interface for each subprogram in this "main" file. Do not use `contains`.

Read the file name from the command line. Reuse your subroutine to read the file but you will need to add some things. First of all you will need to count the number of lines in the file. Write a function count_lines that does this and returns the number. It is up to you whether you pass it the number of header/footer lines or whether you will apply that information later. Your read_data routine will then need an interface for the count_lines function. Count_lines can check for the existence of the file and return 0 if it is not found.

Still in read_data, using the number of items in the file, corrected for the header and the two footers, allocate an array of bird_data types. Loop through this array calling your constructor for each species. The read_data routine should return the array of years and the array of bird_data types.

Request a species name from the user. Find the species in your array of types and print its mean, standard deviation, and results from minmax.

Compute an array of the means for all species. Use the pshellsort routine from sorters to sort this array. This procedure also returns the *permutation vector*, which is an array of the indices of the original positions. For example, if after the sort the permutation vector is (17,3,55,11,23, and so forth) that means that the element that was previously 17 is now the first in the new array, and so on. Note that these sorters return in ascending order (smallest to largest).

From the sorted mean array and the permutation index, print the names of the 10 most common (by mean) species over the years of observations. Hint: you can use a trick to reverse a dimension of an array in Fortran: R=A(ndim:1:-1)

Test the user input portion for

TurkeyVulture
TuftedTitmouse
ElegantTrogon

If you have time, make the small changes required to convert the type into a class.

Project 1B (For Python Programmers).

Download the NetCDF files `sresa1b_ncar_ccsm3_0_run1_200001.nc` and `RASM_example_data.nc` from Resources/Data.

Go through the tutorial at

http://nbviewer.jupyter.org/github/nicolasfauchereau/metocean/blob/master/notebooks/xray.ipynb

They call it xray which was the previous name. If you wish you can import it under that name so you don't get confused. You can use iPython or Jupyter to play with the data but when you understand it, type it into a file and run it through Spyder. Open the `sresa1b_ncar_ccsm3_0_run1_200001.nc` file. If you get an error message that makes a suggestion, follow it. Examine the data. Figure out the sizes/shapes of the data. Make a pretty plot of the mean surface temperature over the dataset. You may use the function from the tutorial but be sure to credit it. This dataset has only one time so you can't do the time analyses in the tutorial.

When you have finished that, open the `RASM_example_data.nc` dataset and follow the Example at http://xarray.pydata.org/en/stable/index.html

Project 2 (Either Language)

A. Download the file wm1deg.grid from the Collab site for this assignment (Resources/Data). This is a topographic dataset for the entire Earth at a resolution of 1 deg by 1 deg. Each row is a line of latitude; the entries in the row are the elevations at each longitude along that value of latitude. Elevations are represented as deviations from sea level (plus or minus) in meters, to the nearest meter (i.e. no fractional parts). The starting latitude is 89.5 N (positive) and the starting longitude is 0.5 E. Remember that latitudes run from -90 to 90 while longitudes run either -180 to 180 or 0 to 360.

Your program should read the name of the topographic dataset from the command line. Do not hardcode it or ask for user input.

Python: Use loadtxt to read in the data and use Matplotlib to make a contour plot of it. The standard for maps is that north is at the top of the page and east is to the right. Does your picture conform to this? You will need to make one-dimensional arrays for lat and long that are of the appropriate size and whose values you compute from the starting points and the resolution. You can use these as the x and y arrays for your contour plots.

Fortran: Create a namelist file and use it to read in the parameters for latitude and longitude (start value and resolution for each) as well as the number of values in each dimension. Use that information to set up your arrays and read in the file (i.e. it is not necessary to count lines and rewind). The 1deg file has 180 values in latitude and 360 values in longitude.

Find the locations of the maximum and minimum elevations. Python: plot these points on your final contour plot. Are they where you think they should be? Both:

print out the latitude and longitude for the max and min as well as the corresponding elevation.

2. write a function to compute the total surface area of the Earth that satisfies some criterion. The element of area of the surface of a sphere is given by $dA=R^2\cos(\theta)d\theta d\phi$. (Cosine rather than sine because we measure latitudes from the equator and not the pole.) The radius of the Earth is 6378.1 km. Theta θ is latitude and phi ϕ is longitude. To compute the area, we sum up all the (approximate) area elements. The $d\theta$ is given by the latitudinal resolution converted to radians and the $d\phi$ by the longitudinal resolution in radians. Remember that latitude must be in radians when you compute the cosine. NumPy has a function for this purpose, as does the math module. This function should take as its arguments the latitude array, the latitude resolution, the longitude resolution, and a mask array that describes the criterion. You can check your code by ignoring topographic information and making sure you get 4π (for $R=1$). Watch your units.

Use this function to compute the fraction of the surface area of the Earth that is ocean.

3. Write another function to compute the total volume that satisfies some criterion specified by a mask array. This is very similar to computing surface area but now we must use topographical information as well for the third dimension, which is the length of the column (positive, please). The volume element to be summed is $dV=zdA$.

Use this function to compute the volume of the ocean. You may find that your fraction won't be all that accurate but it shouldn't be wacky, so be sure to sanity-check your result. (It's surprisingly close considering the coarse resolution.) Print your results with some explanation of what each number is. Watch your units.

You should put your subprograms that are related to geographical manipulations or computations into a module called geo.py. Functions related to reading and displaying data may go into your main program or you may create a module for that.

B. Python: Use Basemap to add coastline figures to your contour plot of the topography.

C. If you still have time, write a function that extracts a specified contiguous region from the data. NumPy has functions to help with this, such as nonzero. As an application, the boundaries of North America are, very roughly, 12 degrees N to 66.5 degrees N and 130 degrees W to 60 degrees W. Don't forget to convert the longitude to whatever numerical system you are using. Pull out the data for this rectangle. Python: make a contour map of it. Add geographical and political boundaries with Basemap. Fortran: print it out in an appropriate format.