Chapter 11

Berkeley Earth and data in text files

Berkeley Earth[1] is a project that provides an independent reconstruction of Earth's recent temperature history. It took several sets of historical weather station observations and used them to perform the reconstruction. All data that Berkeley Earth used, both raw and transformed (filtered for bad data, adjusted for systematics etc.) are freely available from the project's website. We will be using Berkeley Earth's data for this exercise. We will not perform careful analysis, but use the data to gain experience reading and writing textual data sets.
In this exercise we will:

- read and write files,
- create loops, and
- learn to use the basic Python data types.

11.1 Assignment

For this exercise you cannot use any modules not in the Python standard library, and furthermore you are also not allowed to use the `csv` module that is available from the Python standard library. Start a report (A4, PDF) that you use to explain the steps you took. This report is to be handed in along with the Python scripts you will create, use it to answer questions below and if needed to explain your methods.

11.1.1 Reading data

The first step is to write some Python code to process a single file of Berkeley Earth data.

- Go to the data section of Berkeley Earth's website and read the data overview before doing anything else.
- Download the monthly average temperatures for land only. You will write a simple Python program to read these data. Open the data file in a text editor and have a look at its content. You will see that there is a header containing some meta data and after that columns of data. Note that the data file contains the temperature anomaly, not the actual temperature.
- Create a new Python script called `datareader.py`. Be sure to add a comment, i.e. a line starting with a `#`, with your name and student number to the file — Python will ignore this line. Make sure that the data file is in the same directory as you Python program (for convenience). To confirm you can execute the program make it print hello world to the terminal.
- Open the data file using the `open` function as described in Chapter 7.
- Now open the data file and iterate over its contents while printing each line to the terminal.

[1] See `http://berkeleyearth.org/`.

- You are now ready to write a loop that extracts the actual data from the data file. The format for the data that you should make is a list of tuples, where each tuple contains the year, the month, the temperature anomaly and its uncertainty.
- For your report: which data types in Python are appropriate to represent the aforementioned values?
- Write a loop that extracts the aformentioned values and creates the list of tuples containing the values. Be sure to chose an appropriate value to represent missing data (the NaN values in the data file).
- For extra credit: extract the January 1951 - December 1980 temperature values from the file and add them to the temperature anomalies you extracted before to arrive at real temperatures (all using Python and an unmodified data file).
- For your report: which month and year has the largest temperature anomaly and which the smallest (write some code to do this)? Report also the values of either the temperature anomaly or the temperature.
- To be handed in: Your Python code saved as a file `datareader.py`.

11.1.2 Refactoring

Choose any 4 weather stations with at least 250 measurements each available from the Berkeley Earth data set. You will now write a small program to process these files in such a way that one program can process all 4 files.

- Create a new Python script and as before create a loop to extract the year, month, temperature anomaly (or temperature anomaly + average) and uncertainty. Do this for one data file.
- We are now going to use the template provided in Section 11.1.5 to write a simple program that can process each of the data files you chose. You should copy the template to a file called `stationdata.py`. The template contains a function definition (the line starting with `def`) that defines a function `read_file` that accepts one parameter, the name of the file it operates on) that you can use as is. Before changing program confirm that it runs without crashing.
- Adapt your code so that it works inside of the function `read_file`. If you did this correctly the program should print out how many data entries each file has.

The process of re-organizing code to be better organized or a better fit to the problem at hand is called *refactoring*. When you write larger programs you will often have to refactor parts of them. In this case you made a piece of code more general, going from being able to process only one file, to being able to process an arbitary number of files — assuming their contents are laid out in the same way. Next week we will treat function definitions and program structure in more detail.

11.1.3 Writing data

You will now write a small program to write the anual average temperature anomalies to a text file. You can re-use parts of the programs you wrote earlier.

- Create a new Python file called `anualaverages.py`.
- Note that for this exercise you do not need to define functions yourself yet — a “structureless” program is ok for now.
- Copy the file reading code from either of the previous exercises and extend it with a way of calculating anual averages. Make sure that you take into account months that have missing data properly in your averages.
- Make sure that the format of the data you produce is a list of tuples where each tuple contains the year, temperature (anomaly) and uncertainty on that temperature (anomaly) represented by appropriate data values.

- Now add code to open a file for writing, and loop over your anual averages. For each entry write a line to the file with a column for the year, a column for the temperature (anomaly) and a column for the uncertainty on that temperature (anomaly). The columns should be separated by white space (a tab or a space).
- You are now almost ready, but the data file you wrote has no header describing what it contains. Add a code to write a line explaining what each column contains. These also should be white space separated and easy to read with by a program.

11.1.4 Hints

- To represent a newline Python uses the `\n` escape sequence, lines accessed in a file will end with this character on UNIX. The `'\n'` character is also called *line feed*. On Windows the end of lines need `'\r\n'`, a combination of *carriage return* and line feed. You will need to strip these characters when processing the files (look at the `strip()` method of strings).
- Look up the `split`, `join`, `startswith` and `endswith` methods of strings on the Python website — you will need them.
- Look up the `min` and `max` functions.

11.1.5 Program template

```
#!/usr/bin/env python
'''
Small program to read single weather station data from Berkeley Earth's data set.
'''


def read_file(filename):
    '''
    Read a single data file given by filename.
    '''
    data = []  # prepare an empty list that you will fill
    # --------------------------------
    # Add your file handling code here:

    # --------------------------------
    return data  # return the data to

if __name__ == '__main__':
    filenames = [
        'YOUR FIRST FILENAME HERE',
        'YOUR SECOND FILENAME HERE',
        'YOUR THIRD FILENAME HERE',
        'YOUR FOURTH FILENAME HERE',
    ]

    for filename in filenames:
        data = read_file(filename)
        # You need not understand the following line for now:
        print 'File %s contains %d data entries' % (filename, len(data))
```

11.2 Submitting

The submission for this week should contain:

- A report (A4, PDF) called `week3.pdf`
- A Python program called `datareader.py`
- A Python program called `stationdata.py`
- A Python program called `anualaverages.py`
- A `gzip`-ed tar ball `stationdata.tar.gz` with the data file you used for the second program.