Chapter 13

Simple plotting

The goals for this week are to:

- apply the rules for program structure in Python,
- produce some basic plots using `matplotlib`.

This exercise will be graded as follows, you can earn at most 9 points by completing the exercise and handing in on time. Your grade g, awarded on a scale from 1 to 10, is calculated as follows:$g = 1 + s$, where s are the points you earned. For this exercise the points are awarded for the following things:

(3 points) Style: proper naming scheme, documentation,

(3 points) For completing Section 13.1 correctly.

(3 points) For completing Section 13.2 correctly.

13.1 Assignment part 1

This part of the exercise is a warm up of what is to follow. For this part of the exercise you will create a parser for a climate reconstruction by the Berkeley Earth project.

- Create a report for this week, where you can add the plots you created. As always you should use a A4 sized PDF file.
- Download the data available from `http://berkeleyearth.lbl.gov/auto/Global/Land_and_Ocean_complete.txt` and save it to a file called `Land_and_Ocean_complete.txt`.
- As you can see by opening the text file in a text editor, the structure of the file is much simpler than that of the KNMI data of last week. Note that there are two data sets in the file, read only the first.
- Starting from the scaffold provided below, write a program to read the data and plot the monthly average temperature anomaly. See the points below for the steps this entails.
- Fill the function body of `read_data` with code that opens the file, skips the header in that file, reads in all the data (i.e. iterates over all lines and converts the columns to appropriate data types) and returns the data that was read. You can represent the missing values as not-a-number: `float('nan')`.
- Fill the function body of `plot_data` with code that makes two lists, one for the months since january 1850 (you are allowed to just count them from 0), and a list for the monthly temperature anomaly. You can now use the `plot` function from `pyplot` to create a plot of the monthly temperature anomaly since January 1850.

- Make sure to properly label the plot you create before handing in.
- Make sure to add docstrings to your program.
- Make sure that your program runs correctly when the data file is placed in the same directory.
- Add the plot you created to your report.

```
#!/usr/bin/env python
from matplotlib import pyplot


def read_data(filename):
    pass


def plot_data(data):
    pass


def main(filename):
    data = read_data(filename)
    plot_data(data)


if __name__ == '__main__':
    main('Land_and_Ocean_complete.txt')
```

13.2 Assignment part 2

In this part of the exercise you will fix and refactor the script you wrote last week to read the KNMI data and plot it. In this part of the exercise you are allowed to skip reading the header and just use your knowledge about the different columns in the data.

- Below a program scaffold is provided that you can use to fix the KNMI data file parser of last week. Note that the third part of the header is already parsed by the program scaffold.
- Using your own code of last week and optionally the provided scaffold, build a parser for the KNMI data file of last week. This time around we are not modifying the data file. Note that the filename `chunk.txt` is a placeholder for the full data file.
- Add a function body to the `plot_max_temperature` that plots the maximum temperature at a certain weather station (identified by station number).
- Make sure to properly label the plot you create before handing in.
- Make sure to add docstrings to your program.
- Make sure that your program runs correctly when the data file is placed in the same directory.
- Add the plot you created to your report.

```python
#!/usr/env/bin python


def read_station_locations(file_obj):

    # look for start of the first part of the header
    for line in file_obj:
        if line.startswith('# STN'):
            break

    # parse the station locations until an empty line is reached
    station_locations = []
    for line in file_obj:
        if line.startswith('#') and not line[1:].strip():
            break

        # parsing code goes here:
        print 'STATION LOCATION', line

    return station_locations


def read_abbreviations(file_obj):
    # parse abbreviations until an empty line is reached
    abbreviations = {}
    for line in file_obj:
        if line.startswith('#') and not line[1:].strip():
            break

        # parsing code goes here
        print 'ABBREVIATOIN', line

    return abbreviations


def read_column_names(file_obj):
    """Read column names from KNMI data file"""
    column_names = [cn.strip() for cn in next(file_obj).split(',')[1:]]

    return column_names


def read_data(file_obj):
    """Read the weather observations from the KNMI data file"""
    pass  # parse the body of the data file here


def read_knmi_data_file(filename):
    """Fully parse the KNMI data file"""
    with open(filename, 'r') as f:
        station_locations = read_station_locations(f)
        abbreviations = read_abbreviations(f)
        column_names = read_column_names(f)
        data = read_data(f)

    return station_locations, abbreviations, column_names, data


def plot_max_temperature(data, station_number):
    """Given its number plot max temperature over time at a weather station"""
    pass


def main():
    """Parse the KNMI data file and plot the max temperature at the Bilt."""
    station_locations, abbreviations, column_names, data = \
        read_knmi_data_file('chunk.txt')
    plot_max_temperature(data, station_number=260)


if __name__ == '__main__':
    main()
```