UNIVERSITY OF AMSTERDAM

ANTON PANNEKOEK INSTITUUT

Basic Linux and Coding for AA (BLAC)
Exercise 4 (week 2)

Author:
Timo Halbesma, 6126561

Supervisor:
Dr. T. Coenen

Listing 1: TLRH's solution for the BLAC homework 3 (week 2) . Not that this code has been slightly modified since my last submission.

```
# knmi-1-6126561.py <-- Assignment 2

# Python script for Basic Linux and Coding for AA homework 3 (week 2).
# Usage: python knmi-1-6126561.py
# TLR Halbesma, 6126561, september 9, 2014. Version 1.0; implemented

# NB All functions in this program require the entire dataset as input.
# This behavior could be altered such that main() subsets the dataset and feeds
# it to the functions. I might change this later on for aesthetic reasons.

# Not yet required. However, dr. Coenen did mention we will plot this data.
# import math
# import matplotlib.pyplot as plt

INPUTFILE = './KNMI_20000101.txt'

def read_data(datasetKNMI, endLine): # <-- Assignment 5
    """
    Function to read KNMI dataset obtained from
    http://www.knmi.nl/climatology/daily_data/selection.cgi

    datasetKNMI : list containing the entire dataset including header

    returns a list containing a list of all datapoints per station per date.
    """

    lines = []

    # Assignment 3
    # Skip first 85 lines because that is the header. Very ugly solution :-(
    # NB this breaks down if the header size changed. Be cautious!
    for i in range(85,endLine): # Header: first 85 lines. Read endLine lines.
            myLine = datasetKNMI[i].strip().split(',') # strip to remove '\n'
            cleanLine = []
            for entry in myLine:
                # entry.strip() removes the whitespace around the datapoint.
                # entry.strip() returns False if len(x.strip()) == 0 (missing..)
                if entry.strip():
                    cleanLine.append(int(entry.strip()))
                else:
                    # Assignment 4. Use None for missing data entries.
                    cleanLine.append(None)
            lines.append(cleanLine)

    return lines

def read_StationID(datasetKNMI): # <-- Assignment 8
    """
    Function to read header from KNMI dataset, in particular the station info.

    datasetKNMI : list containing the entire dataset including header.

    returns a list containing one list for each station.
    """

    allStations = datasetKNMI[3:41]
    allStationsCleaned = list()

    for station in allStations[1:]: # First line contains column info, remove.
        # Remove leading '#', split and unpack first four columns.
        stationID, lon, lat, alt = \
                station.replace(':','').strip('#').split()[:4]
        # The name may contain spaces. Take sublist until last element.
        name = ' '.join(station.strip('#').split()[4:])

        # Create list containing one list for each station.
        # That list contains for each station 5 entries
        # Expliciet typecasts to sensible datatypes. Trivial datatype choices.
        # StationID is a natural number, thus, an integer.
        # longitude, latitude and altitude are rational numbers, thus, floats.
        # The name consists of multiple characters, thus, is saved to string.
        allStationsCleaned.append([int(stationID), float(lon), \
                float(lat), float(alt), str(name)])

    return allStationsCleaned

def read_ColumnDescription(datasetKNMI): # <-- Assignment 9
```

```
    """
    Function to read header from KNMI dataset, in particular column descriptions

    datasetKNMI : list containing the entire dataset including header.

    returns a dictionary mapping the column name to its description
    NB dictionaries may be printed in random order.
    """

    columnDescription = datasetKNMI[42:82]
    columnDescriptionCleaned = dict()

    for entry in columnDescription:
        abbreviation = ''.join(entry.strip('#').split('=')[:1]).strip()
        description = ' '.join(entry.strip('#').split('=')[1:])
        columnDescriptionCleaned[abbreviation] = description

    return columnDescriptionCleaned

def read_ColumnHeader(datasetKNMI): # <-- Assignment 10
    """
    Function to read header from KNMI dataset, in particular column header.

    datasetKNMI : list containing the entire dataset including header.

    returns list of column names.
    """

    columnHeader = datasetKNMI[83:84]

    return ''.join(columnHeader).strip('#').strip().replace(' ', '').split(',')

def main():
    # Assignment 1
    f = open(INPUTFILE, 'r')
    datasetKNMI = f.readlines()
    f.close()

    # Assignment 12
    print read_data(datasetKNMI, 500)[0], '\n\n' # Read until line 500.
    print read_StationID(datasetKNMI), '\n\n'
    print read_ColumnDescription(datasetKNMI), '\n\n'
    print read_ColumnHeader(datasetKNMI)
    # NB there is one entry more in the list returned by read_ColumnHeader
    # STN is in the line with column headers but it has no description.

# This codeblock is executed from CLI, but not upon import.
if __name__ == '__main__': # <-- Assignment 6; was already in my file though.
    main()
```

Listing 2: TLRH's solution for the BLAC homework 4 (week 2)

```
# -* coding: utf-8 -*
#!/usr/bin/python
# knmi-6126561.py <-- Step 1

# Python script for Basic Linux and Coding for AA homework 4 (week 2).
# Usage: python knmi-6126561.py
# TLR Halbesma, 6126561, september 12, 2014. Version 1.0; implemented

import math
import matplotlib.pyplot as plt
import numpy as np

# Import methods and variables from homework 3 (week 2).
from knmi_1_6126561 import * # <-- Step 2

# Override INPUTFILE with dataset that does not include 20000101!
# NB this is a slightly different dataset than I used for the prior assignment
INPUTFILE = './KNMI_19991231.txt'

# Make data available troughout all methods.
knmiData = list()
knmiStationIDs = list()
knmiColumnDescription = dict()
knmiColumnHeader = list()

def readDataset():
    f = open(INPUTFILE, 'r')
    datasetKNMI = f.readlines()
    f.close()

    global knmiData
    global knmiStationIDs
    global knmiColumnDescription
    global knmiColumnHeader

    # Obtain data and entries using last homework3's methods.
    knmiData = read_data(datasetKNMI, len(datasetKNMI)) # Read all lines.
    knmiStationIDs = read_StationID(datasetKNMI)
    knmiColumnDescription = read_ColumnDescription(datasetKNMI)
    knmiColumnHeader = read_ColumnHeader(datasetKNMI)

    print "readDataset successful"

# Using ColumnDescription, find string s in the values. Then find the index of
# values' key in ColumnHeader. This integer is needed to know what column to
# sort in in the knmiData. e.g. 'Maximum temperature', 'precipitation', etc.
def findColumnName(myIdentifier):
    ColumnAbbreviation = None
    # Loop trough ColumnDescription, find given string in value (description).
    for key,value in knmiColumnDescription.items():
        if myIdentifier in value:
            # Now get the key (abbreviation) and find it in the ColumnHeader.
            ColumnAbbreviation = key
            break
    if ColumnAbbreviation: # Check if ColumnAbbreviation is found.
        return knmiColumnHeader.index(ColumnAbbreviation)
    else:
        return None

def findStationName(myStationID):
    for station in knmiStationIDs:
        if station[0] == myStationID:
            return station[-1]

def findMax(myDataSet, columnNumber, toReverse): # <-- Step 3
    """
    Find the maximum value in the data set given a columnNumber to sort on.
    Found sorting a matrix on http://xahlee.info/perl-python/sort_list.html

    myDataSet : nested list. Contains the dataset that should be sorted.
    columnNumber : int. Specify which column should be sorted on.
    toReverse : boolean. True => reverse (max -> min); False => (min -> max)

    returns a list containing the entry of the max (or min) in the dataset.
    """

    myDataSet.sort(key=lambda x:x[columnNumber], reverse=toReverse)
    return myDataSet[0]

    # Implemented because the built-in sort function did not behave well.
    # It turned out I stored my dataentries as str instead of int, thus,
    # sort broke down, just as this implementation failed.
    #theMax, maxEntry = 0, []
    #for entry in myDataSet:
    #    if entry[columnNumber] > theMax:
    #        print entry
    #        print entry[columnNumber]
    #        theMax = entry[columnNumber]
    #        print type(theMax)
    #        maxEntry = entry

def seriesOfMinMaxPrecipitation(myDataSet, stationID): # <-- Step 4
    # station of choice in 1968
    choice1968 = list()
    precipitationNumber = findColumnName('precipitation amount')
    hottestNumber = findColumnName('Maximum temperature')
    coldestNumber = findColumnName('Minimum temperature')

    for entry in myDataSet:
        # entry[1] is the date YYYYMMDD as integer. So div by 1e5 will
        # result in YYYY. As it is int-int division it is truncated.
        if entry[0] == stationID and entry[1]/10000 == 1968:
            choice1968.append(entry)

    # This part fully depends on the assumptions what to include in the time
    # series. But one can find a time-series of max/min/precipitation
    for entry in choice1968:
        continue
        #entry[precipitationNumber]
        #entry[hottestNumber]
        #entry[coldestNumber]

    plt.clf()
    plt.plot([x for x in range(len(choice1968))], \
            [y[precipitationNumber]/10.0 for y in choice1968], lw=1)
    plt.xlabel('Number of the day in 1968',fontsize=16)
    plt.ylabel('Precipitation amount in mm')
    plt.title('precipitation amount for '+str(stationID)+' in 1968.')
    plt.show()

    plt.plot([x for x in range(len(choice1968))], \
            [y[hottestNumber]/10.0 for y in choice1968], lw=1)
    plt.xlabel('Number of the day in 1968',fontsize=16)
    plt.ylabel('Maximum temperature in degrees Centigrade.')
    plt.title('Maximum temperature for '+str(stationID)+' in 1968.')
    plt.show()

    plt.plot([x for x in range(len(choice1968))], \
            [y[coldestNumber]/10.0 for y in choice1968], lw=1)
    plt.xlabel('Number of the day in 1968',fontsize=16)
    plt.ylabel('Minimum temperature in degrees Centigrade.')
    plt.title('Minimum temperature for '+str(stationID)+' in 1968.')
    plt.show()

    #return choice1968

# Step 5
# Compare the summers in  De kooy  with those in  Valkenburg . Calculate
# monthly averages for min, max temperature and the amount of precipitation
# on a 10 yearly basis. Where are the summers warmer, where are they
# wetter?

def plotComparison(valkenburgData, deKooyData, s):
    # http://matplotlib.org/examples/api/barchart_demo.html
    title = {'TX': 'maximum temperature', 'TN':'minimum temperature',\
            'RH': 'daily precipitation'}
    monthNames = ['Jan', 'Feb', 'Mrt', 'Apr', 'Mei', 'Jun', 'Jul', 'Aug',\
            'Sep', 'Okt', 'Nov', 'Dec']
    ind = np.arange(1,13)
    width = 0.35
    fig, ax = plt.subplots()
    rects1 = ax.bar(ind, tuple([valkenburgData.get((s,i, 5))\
            for i in range(1,13)]), width, color='r')
    rects2 = ax.bar(ind+width, tuple([deKooyData.get((s,i, 5))\
            for i in range(1,13)]), width, color='y')

    ax.legend((rects1[0], rects2[0]), ('Valkenburg', 'DeKooy'))
    ax.set_ylabel(s)
    ax.set_title('Plot of '+title[s])
    plt.xticks(range(1,13), monthNames, rotation=45)
    plt.show()
    #plt.close()

def compareDeKooyValkenburg(myDataSet):
    # NB this function can be generalized as a function of StationID
    # such that it returns the 10yr per-month averages.
    valkenburg, valkenburgData = 210, dict()
    deKooy, deKooyData = 235, dict()

    # Counters for the number of entries in the dataset to divide sum by.
    valkenburgTX, valkenburgTN, valkenburgRH = dict(), dict(), dict()
    deKooyTX, deKooyTN, deKooyRH = dict(), dict(), dict()
    for i in range(10):
        valkenburgTX[i], valkenburgTN[i], valkenburgRH[i] = int(), int(), int()
        deKooyTX[i], deKooyTN[i], deKooyRH[i] = int(), int(), int()

    # Find the column numbers
    precipitationNumber = findColumnName('precipitation amount')
    hottestNumber = findColumnName('Maximum temperature')
    coldestNumber = findColumnName('Minimum temperature')

    # Set the dictionary values to zero to sum in.
    for i in range(1,13):
        for j in range(5, 10):
            valkenburgData[('TX',i,j)] = int()
            valkenburgData[('TN',i,j)] = int()
            valkenburgData[('RH',i,j)] = int()
            deKooyData[('TX',i,j)] = int()
            deKooyData[('TN',i,j)] = int()
            deKooyData[('RH',i,j)] = int()

    for entry in myDataSet:
        if entry[0] == valkenburg:
            # entry[1] is the date YYYYMMDD as integer. So (div by 100)%100
            # will result in MM. As it is int-int division it is truncated.
            month = (entry[1]/100)%100
            # split year up in blocks of 10. Works because year in [1950-2000]
            # Note that the dataset must not include 2000!!
            year = (entry[1]/100000)%10
            if entry[hottestNumber]: # Missing data has value None in dataset.
                valkenburgTX[year] += 1
                valkenburgData[('TX',month,year)] += entry[hottestNumber]
            if entry[coldestNumber]:
                valkenburgTN[year] += 1
                valkenburgData[('TN',month, year)] += entry[coldestNumber]
            if entry[precipitationNumber]:
                valkenburgRH[year] += 1
                valkenburgData[('RH',month,year)] += entry[precipitationNumber]
        elif entry[0] == deKooy:
            month = (entry[1]/100)%100
            year = (entry[1]/100000)%10
            if entry[hottestNumber]:
                deKooyTX[year] += 1
                deKooyData[('TX',month,year)] += entry[hottestNumber]
            if entry[coldestNumber]:
                deKooyTN[year] += 1
                deKooyData[('TN',month,year)] += entry[coldestNumber]
            if entry[precipitationNumber]:
                deKooyRH[year] += 1
                deKooyData[('RH',month,year)] += entry[precipitationNumber]

    # Now divide the per-month 10yr sums over the number of entries.
    for i in range(1,13):
        for j in range(5, 10):
            if valkenburgTX[j] != 0:
                valkenburgData[('TX',i,j)] /= float(valkenburgTX[j])
            if valkenburgTN[j] != 0:
                valkenburgData[('TN',i,j)] /= float(valkenburgTN[j])
            if valkenburgRH[j] != 0:
                valkenburgData[('RH',i,j)] /= float(valkenburgRH[j])
            if deKooyTX[j] != 0:
                deKooyData[('TX',i,j)] /= float(deKooyTX[j])
            if deKooyTN[j]:
                deKooyData[('TN',i,j)] /= float(deKooyTN[j])
            if deKooyRH[j]:
                deKooyData[('RH',i,j)] /= float(deKooyRH[j])

    plotComparison(valkenburgData, deKooyData, 'TX')
    plotComparison(valkenburgData, deKooyData, 'TN')
    plotComparison(valkenburgData, deKooyData, 'RH')

    return valkenburgData, deKooyData


# Step 6
# Using the monthly averages (averaged over 10 year blocks), is the weather
# getting warmer or wetter?
def warmerOrWetter():
    # To implement this function requires rewriting the very crappy
    # implementation of step 5.
    return None

def main():
    readDataset()

    precipitationNumber = findColumnName('precipitation amount')
    wettestDay = findMax(knmiData, precipitationNumber,  True)
    print "The wettest day was at {0} in {1}({2}).".format(\
            wettestDay[1],findStationName(wettestDay[0]),wettestDay[0]),
    print "The precipitation amount was {} mm."\
            .format(wettestDay[precipitationNumber]/10.0)


    hottestNumber = findColumnName('Maximum temperature')
    hottestDay = findMax(knmiData, hottestNumber, True)
    print "The hottest day was at {0} in {1}({2}).".format(\
            hottestDay[1],findStationName(hottestDay[0]),hottestDay[0]),
    print "The temperature was {} degrees Centigrade."\
            .format(hottestDay[hottestNumber]/10.0)

   seriesOfMinMaxPrecipitation(knmiData, 260)

    valkenburg, deKooy =compareDeKooyValkenburg(knmiData)
    print 'valkenburg'
    for k,v in valkenburg.items():
        print k,v
    print 'deKooy'
    for k,v in deKooy.items():
        print k,v

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