CSci 1133
Lab Exercise 10

## File I/O

Large datasets are the domain of computing. Although short "script-like" programs are often needed, more general computational tasks involve processing large volumes of data. Python provides several powerful constructs for reading data from, and writing data to, external storage media.

# Warm-up

### Primer on CSV File Format

We've learned that text files are simply long strings (sequences of characters) that have been stored on some non-volatile media. Text files can contain any data that is representable using characters: e.g., textbooks, novels, computer program source code, lab data, etc. Although there is no standard text file "format", a universally accepted format has emerged to store tabular data (i.e., spreadsheet data) as a text file. This standard is called the "Comma Separated Values" format, or CSV for short.

CSV format is quite simple in practice: individual rows of a table or matrix are delimited using the *newline* character, '\n', and individual values within a row (column values) are delimited using *commas*. For example, consider the following "tabular" (matrix) values:

```
1    2    3

4    5    6

7    8    9
```

If the matrix is stored in CSV format, the resulting text file would contain the characters:

```
"1,2,3\n4,5,6\n7,8,9"
```

and, using any standard text editor to display the file contents, we would see:

```
1,2,3
4,5,6
7,8,9
```

Each "row" of a CSV file is referred to as a file *record*. And the data elements in each row are referred to as "fields". In the matrix example above there are three records, each consisting of three fields.

### Writing CSV files

Since the `.write` method takes a *single* string argument, we have to construct each record of the file in the same way we constructed lists. Since tabular data is two dimensional, this usually suggests a nested loop. Here is one way in which we might write the matrix to a CSV formatted text file:

```
fileobj = open('somefile','w')
for i in range(3):
    record = ''                       # start with a null string
    for j in range(1,4):
        record += str(i*3+j) + ','    # append each value and comma
    record = record[:-1]              # strip off the last comma
```

```
        fileobj.write(record)
        if i < 2:
            fileobj.write('\n')          # no \n on last record!
    fileobj.close()
```

Any tabular data can be represented in CSV format and it is a common method to export Excel Spreadsheet data to be subsequently processed using a computer program.

1) **Generating Synthetic Test Data**

Write a Python program that will prompt the user for the name of a file and create a CSV (comma separated value) file with 1000 *lines* of data. Each line will contain 2 values: the line number (starting with 1) and a randomly generated integer value in the closed interval [-1000, 1000].

Program constraints: do not import/use the Python `csv` module.

**Warning!** If you accidentally construct an infinite loop, you will most likely exhaust your CSELabs storage quota before you are able to stop the program and your account will then be suspended. Therefore, we suggest that you substitute a `print()` statement in place of `.write()` when you begin testing your program. Replace the print statement with `.write()` *after* you are comfortable that your program is working!

After you've created the file, use a text editor to open it and examine its contents to verify that it was created correctly.

## Stretch

1) **Letter Frequencies**

A number of computer applications make use of non-linear distributions of data values for various things like efficiently compressing large volumes of data or decoding encrypted messages. It is generally accepted that the individual letter 'e' is the most frequently occurring letter in the English language. In this problem we will explore the frequency of letters in a large corpus of English text and confirm if this is the case.

Use your web browser to view the following URL:

`http://textfiles.com/directory.html`

This is a website that maintains a large collection of English text corpora. Browse the site and download a suitably large (> 100,000 bytes) text file. Write a python program that will process this file and produce a sorted letter frequency listing. Do not import/use the Python `csv` module.

Generate the letter frequency by counting individual letters in the text. If you are familiar with Python *dictionaries*, then you may use a dictionary for this, otherwise you should use a list of individual integer counters and determine how to convert a character to an offset into the list.

Note: you will need to "clean up" the text to remove spaces, punctuation, whitespace etc. and convert it to all upper or lower case before analyzing the characters.

## Workout

1) **Earthquake Data, part 1**

Large repositories of recorded measurement data are available on the World Wide Web from a wide spectrum of applications such as stock market data, weather/climate data, etc.

Use your Web browser to view the following URL:

```
http://earthquake.usgs.gov/earthquakes/feed/
```

This site is maintained by the US Geological Service and reports recent earthquake activity around the world in real time. Scroll down the page until you see a link titled 'Spreadsheet Format'. Select this link and then scroll down the next page until you locate 'Feeds'. From the 'Past Day' menu, select 'All Earthquates' and then save the downloaded file in your home directory. Open the .csv file with your text editor and examine it.

The file contains a large quantity of information that is detailed in the first line of the file. We will only be interested in the magnitude of the earthquake and the place (the string indicating where the event occurred, *not* the latitude/longitude). As you examine the file, note that the `place` is actually *two* comma separated strings. The first begins with a double-quote and the second ends with another double quote. The "general" location we are interested in is the *second* of the two strings that make up the place description.

Write a Python program that will read the file and display the categories from the first line, along with their relative location in the line (do not import/use the Python `csv` module):

Example:

```
0  time
1  latitude
2  longitude
3  depth
4  mag
5  magType
6  nst
7  gap
8  dmin
9  rms
10 net
11 id
12 updated
13 place
14 type
15 horizontalError
16 depthError
17 magError
18 magNst
19 status
20 locationSource
21 magSource
```

2) **Earthquake Data, part 2**

Now modify your program to print out the *magnitude* and *location* of each earthquake in the file. The *location* is the information that appears in the second "place" field (after the embedded comma). The list should be printed in descending order of magnitude and the "*location*" should be clean of any quotation symbols. Compare your program results to the actual file data to verify that it works correctly.

## Challenge

Here is an interesting challenge problem. Try it if you have extra time or would like additional practice outside of lab.

**Reading Internet Data**

Python provides a useful module named `urllib` that enables reading data from Internet resources such as web pages. The `urllib` module actually consists of several sub-modules which you can identify using the `help( )` function.

```
>>> import urllib
>>> help(urllib)
```

The particular sub-module `urllib.request` provides methods to allow you to construct programs that interact with web-based html resources in much the same way as ordinary text files. In order to read a webpage, you must first open it using the `urlopen()` method that works much like the regular file open but takes a string argument containing the URL of the webpage you want to read:

```
>>> from urllib import request
>>> page =  request.urlopen("http://cs.umn.edu")
```

The URL object is much like a text file. You can use the same methods to read the webpage that you use to read a file. Try reading the first line of the cs.umn.edu webpage:

```
>>> htmltext = page.readline()
>>> htmltext
b'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">\n'
```

Although this looks like a simple string, notice that the "line" contains a subtle difference... it starts with `b'` instead of the usual quotation mark. This is actually a *byte array* and needs to be converted to a string before we try to do the usual string operations:

```
>>> line = htmltext.decode()
>>> line
'<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">\n'
```

1) **Stock Market Data**

There are numerous websites that provide stock market data. A well-known example is the webpage `www.finance.yahoo.com`. One of the services provided by the Yahoo website will produce daily stock information for any particular stock given its NYSE symbol. For example, you can obtain a CSV formatted listing of the daily stock market activity for the local biomedical company Medtronic, Inc. by looking up its NYSE symbol: MDT at `http://ichart.finance.yahoo.com/table.csv`. Let's try it!

Use your web browser to view the following URL:

```
http://ichart.finance.yahoo.com/table.csv?s=MDT
```

The `?s=MDT` is how we provide an argument to the webpage. We can also provide additional arguments by separating them with a '`&`' symbol. For example, to obtain just the data for the month of September, we would use the following:

```
http://ichart.finance.yahoo.com/table.csv?s=MDT&a=8&b=1&c=2014&d=8&e=30&f=2014
```

The additional arguments are as follows:

| | |
|---|---|
| `a=8` | ... starting month-1 |
| `b=1` | ... starting day |
| `c=2014` | ... starting year |
| `d=8` | ... ending month-1 |
| `e=30` | ... ending day |
| `f=2014` | ... ending year |

Write a python program that will request a stock ticker symbol, a beginning date and ending date, then graph the closing prices of the stock over that period using the Turtle graphics module.