Pandas

We've talked a whole bunch about looping and opening files and reading them. So far we've worked with plain-text files. But some files have unusual dialects. One common example of such a file is an Excel file.

You should have downloaded a file called 'Spreadsheet.xls'. This is the same file as we used earlier, but written in Excel format. To use it, we need to parse that file. The library 'pandas' can do this.

We will first import pandas.

```
>>> import pandas
```

Many users have written their own libraries for doing fun things. These users may choose to distribute these functions for general use by the community. Pandas (http://pandas.pydata.org/) is such a library.

When we use a library that we downloaded from the internet, it is not part of Python's base functionality. Therefore, we tell Python that we'd like to use this extra library with the import command.

Now we're going to take our spreadsheet and use Pandas to import it.

```
>>> xl = pandas.ExcelFile('spreadsheet.xls')
>>> xl
<pandas.io.parsers.ExcelFile object at 0x102a39610>
```

This should seem familiar - much like with our initial exercises in opening files, we are referencing the file and not the data within it. So let's liberate that data!

Excel documents are made up of sheets, which are linked documents containing different subsets of data. We're keeping it simple - all our data is in one sheet.

```
>>> xl.sheet_names
[u'Sheet1', u'Sheet2', u'Sheet3']
>>> df = xl.parse('Sheet1')
>>> df.head()
         Site  Observations  Species  Expenditure
0  Lake Creek             4       12          180
1  Los Alamos             3        8          340
2    Big Bend             4       16          280
3    McDonald             5       20          280
4  Balmorrhea             3        3          174
>>>
```

Hey look - our data!

In Pandas, we can do indexing of our data by location or by values. For example, by location:

```
>>> df.ix[0]
Site            Lake Creek
Observations             4
Species                 12
Expenditure            180

```

Above, I have indexed by the location. A single digit in the square brackets tells Pandas we'd like to see all the values for the zeroth row in our data.

If I wanted to see a specific observation in the zeroeth row, say, the third one, I could index like so:

```
>>> df.ix[0,3]
180.0
```

Alternatively, I could choose to view the third column for all rows:

```
>>> df.ix[:,3]
0     180
1     340
2     280
3     280
4     174
Name: Expenditure, dtype: float64

```

Try slicing up your data in different ways. Does everything give you the expected output? What does the dtype keyword mean?

That's all well and good, but we have these nice row and column names! Let's use them!

```
>>> df = xl.parse('Sheet1', index_col = 0,header=0)
>>> df.head
<bound method DataFrame.head of             Observations  Species  Expenditure
Site
Lake Creek             4       12           180
Los Alamos             3        8           340
Big Bend               4       16           280
McDonald               5       20           280
Balmorrhea             3        3           174>
>>> df.ix['Lake Creek']
Observations      4
Species          12
Expenditure     180
Name: Lake Creek, dtype: float64
```

By specifying an index column as the zeroeth column, we are now able to access the data by the name of the site. So, if you had a bunch of Excel files with the same sites, you could, for example, use df.ix to get the values for Lake Creek from each. Nifty!

When we specify the header as the zeroeth row, we get to do fun things like so:

```
>>> df.ix[:,'Observations']
Site
Lake Creek    4
Los Alamos    3
Big Bend      4
McDonald      5
Balmorrhea    3
Name: Observations, dtype: float64
```

In this way, we can break our data down for manipulation. We can use this to build more complex functions:

```
>>> a = df.ix[:,'Observations']
>>> a
Site
Lake Creek    4
Los Alamos    3
Big Bend      4
McDonald      5
Balmorrhea    3
Name: Observations, dtype: float64

```

a behaves like a list, so we can do iterative functions on it:

```
>>> b = 0
>>> for x in a:
...     b = b + int(x)
...
>>> b
19
```

Because Excel is kind of terrible (and writing it out requires another library), you can output your data to a csv like so:

```
>>> outfile = open('output.csv', 'w')
>>> df.to_csv(outfile)
>>> outfile.close()
```