Key for Homework 3

Here is a key of different ways in which you could solve the challenges in lecture 3. Ben has also posted scripts showing some ways of breaking down the data. These will also be helpful to look at.

First we read in the file.

```
with open('homework.csv') as f:
    line_list = [line.strip().split(',') for line in f]
   ...:

print line_list

[['Site', 'Observations', 'Species', 'Expenditure'],
 ['Lake_Creek', '4', '12', '180'],
 ['Los_Alamos', '8', '340'],
 ['Big_Bend', 'a', '6', '280'],
 ['McDonald', '5', '20', '280'],
 ['Balmorrhea', '3', '3', '174']]

```

Let's start with some due diligance: Are we missing any data?

```
for line in line_list[1:]:
        if len(line) == 4:
                print "yay"
        else:
                print "There is something wrong on this line, %s" %line

yay
There is something wrong on this line, ['Los_Alamos', '8', '340']
yay
yay
yay
```

How you solve this is up to you. We'll talk a little about missing data in the future. Maybe you want to open the file and insert a value for the missing value. I inserted a missing data symbol:

```
line_list[2].insert(2, 'NA')

line_list

[['Site', 'Observations', 'Species', 'Expenditure'],
 ['Lake_Creek', '4', '12', '180'],
 ['Los_Alamos', '8', 'NA', '340'],
 ['Big_Bend', 'a', '6', '280'],
 ['McDonald', '5', '20', '280'],
 ['Balmorrhea', '3', '3', '174']]
```

Let's average the Species column, since it's all present:

```
total = 0

for line in line_list[1:]:
    total += int(line[2])

print total
381
#count the length of the list of numbers
num_len = len(line_list[1:])
```

```
avg = float(total/num_len)
print avg
71
```

Now, we write this out!

```
outfile = open('out.txt', 'w')

outfile.write('I averaged some stuff! It came out to %i species. I got this numb
er by opening the file, reading it to memory, looping over the species column an
d averaging it.' %avg + '\n')

outfile.close()
```

Check that column one is made of strings.

```
for line in line_list:
    if type(line[0]) is str:
        print "Good!"
    else:
        print "Uh-oh, this is not a string."
```

```
This works ... But ... Can you see a problem here? Line is a list of strings ... by definition,
these will all pass this test. How can we check that they are non-numeric strings?


The below solution is a nifty programming trick that we'll talk more about in the coming weeks.

```python

for line in line_list:
    try: print float(line[0]) #Attempt to perform this operation. In this case, we will attempt
to turn our string into a float. If we can do                                        #this, it w
ill print to screen.
    except: print "this is a string and cannot be converted to a float!" #If we cannot convert
to float, print a message saying so.
   ....:
```

Try and except will be discussed in coming weeks and are a handy way to test code that you expect will fail without completely halting your code's functionality.

For columns one and two, we want to make sure that we *only* have numbers.

```
for line in line_list[1:]:
    print float(line[1])


4.0
8.0
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-48-7717c310b286> in <module>()
      1 for line in line_list[1:]:
----> 2     print float(line[1])
      3

ValueError: could not convert string to float: a
```

In this case, python reads the first two values as floats, as they are numeric. But it spits an error message when it reaches a. a is a letter and cannot be cast as an integer. If we wanted python to keep running, even when it hits an error value, we could write the code this way:
```

```
for line in line_list[1:]:
    try:
        float(line[1])
        print "this value is a number"
    except:
        print "this is not a float"

this value is a number
this value is a number
this is not a float
this value is a number
this value is a number

```

Checking for uniqueness is a little challenging. For this, I used a set. Aset converts a list into a series of unique values. If any values are duplicated, they are only listed once in the set. If the length of the set (which contains *no* repeats) is shorter than the original list (which contains repeats), we know that there were repeats inthe original. Therefore, not every value in the original is unique.

```
float_list = [float(line[3]) for line in line_list[1:]]
uniq_set = set(float_list)
#We can use casting to transform a list into a set
{174.0, 180.0, 280.0, 340.0}
#Sets are denoted in curly brackets.
len(uniq_set) == len(float_list)
#This type of statement is a boolean. We talked about this in Lesson 2. This ty
pe of statement evaluates the truth of the two mathematical statements on eithe
r side of the ==
False
```