Python: Files, Intro to Iteration

For Loops

- We're covering iteration more completely next week, but for today, I want to introduce the idea so that we can do more interesting things with files

```
# This is pseudocode, NOT Python

for component in iterable_data:
        new_variable = component
        new_variable.method('option A')
        some_function(new_variable)
        save_transformations(new_variable)

# This for loop performs and saves the transformations for all components of the data variable, iterable_data
# Note the colon after the initial for statement, the consistent indentation, and the lack of an “end”. These are particular to Python
```

For loops iterate over components of data

- What that component is will depend on the data type, only sequences are iterable
  - Strings: component is a single character
  - Lists: component is a list element
  - Numbers: N/A, cannot iterate
- You give a variable name (any valid variable name) to the component while declaring the for loop. This is the name that each successive element will take as the loop goes.

An example of string vs. list iteration

```
>>> dna = 'ATTGTCCCCAGATTT'
>>> for nt in dna:
...     print nt
...
A
T
T
G
T
C
C
C
C
A
G
A
T
T
T
```

```
>>> >>> mydata = ['chr1', 10878,
10945, '<DEL>']
>>> for thing in mydata:
...     print thing
...
chr1
10878
10945
<DEL>
>>> for c in mydata[0]:
...     print c
...
c
h
r
1
>>>
```

Files are a special data type

- Files can be read and written to, **generally one line at a time**
- Text readers (like Notepad) take one giant string of data, and it knows to display new lines with the newline character '\n', and it knows to separate elements with delimiters ('\t', ' ', '', ',', ':')
  - Those delimiters are, in order, a Tab, a space, no delimiter, a comma, and a semi-colon
  - Technically, any ASCII character(s) can be a delimiter, but let's stick to these **(more specifically, the '\t')**

```
>>> mydata
'Chr1\t10878\t10945\t<DEL>\nChr1\t10878\t10945\t<DEL>\n'
```

```
Chr1    10878    10945    <DEL>
Chr1    10878    10945    <DEL>
```

Three main ways to open/close a file

- Files are a unique data type in part because they must be opened before they're used and closed before the program is done
- Python recently implemented some changes to lessen the headache of remembering to close your file

```
>>> f = open('tumorvsnormal.csv', 'r')
>>> myread = f.read()
>>> f.close
>>>
>>> # this next one closes the file automatically at the end of the with loop
>>> with open('tumorvsnormal.csv', 'r') as f:
...     myread = f.read()
...
>>> # this last one works differently, in that it reads line by line
>>> for f in open('tumorvsnormal.csv', 'r'):
...     print f
...
```

Files are a special data type

- This means that when you **read** a file with the most basic read method, Python will scan the entire giant string, with all the delimiters and newlines
- Other file methods will read the data differently

```
>>> with open('tumorvsnormal.csv', 'r') as f:
...     myread = f.read()
...
>>> myread
'chr1\t14907\tA\tG\t689\t574\nchr1\t14930\tA\tG\t790\t668\nchr1\t14933\tG\tA\t1009\t443\n'
>>>
>>> with open('tumorvsnormal.csv', 'r') as f:
...     myread = f.readline()
...
>>> myread
'chr1\t14907\tA\tG\t689\t574\n'
```

Files are a special data type

- Like other methods, the **file.read()** method can have other methods attached to it that further process the read data
  - A good one is **file.read.splitlines()** (it makes a list separating by newline characters)
- If this example looks familiar, it's because it was the basis of last week's exercise

```
>>> with open('tumorvsnormal.csv', 'r') as f:
...     myread = f.read().splitlines()
...
>>> myread
['chr1\t14907\tA\tG\t689\t574', 'chr1\t14930\tA\tG\t790\t668',
'chr1\t14933\tG\tA\t1009\t443']
```

Writing to file is a lot like reading to file

- There are a couple of important differences between reading and writing a file:
  - The 'r' is replaced by 'w'
  - It's f.write() instead of f.read()
  - You definitely should write one line at a time
  - **This time, you need to make sure the delimiters and newlines are in place before you write, so that your output is readable by a text editor**
    - For this, use the 'delim'.join(list) method, if your writable input is a list (it should be)

Writing to file is a lot like reading to file

```
>>> >>> tdata
[['chr1', '14907', 'A', 'G', '689', '574'], ['chr1', '14930', 'A', 'G', '790', '668'],
['chr1', '14933', 'G', 'A', '1009', '443']]
>>> with open('test.txt', 'w') as f:
...     for td in tdata:
...             f.write('\t'.join(td)+'\n')
>>>
>>> # You should get the exact same data as in tumorvsnormal.csv
```

Last week's problem

- Problem 1: Create an empty list and name it **tdata**. An empty list is two brackets with nothing in between (Python ignores spaces)
- Problem 2: Assign the first string of **data** to a variable called **temp**. Using one specific method mentioned earlier, turn the string with the '\t' delimiters into a list. '\t' is the programming shorthand for a Tab. Remember to assign what you do to a variable.
- Problem 3: Append that to your empty list. Do not assign this one to a variable because appending changes things permanently.
- Problem 4: Do this for the other two list items.

This week's problem

- Problem 1: Open the file 'tumorvsnormal.csv' from last week, but this time using one of the methods you learned today
- Problem 2: Replicate the steps we took last week, but use a for loop instead of doing it three times
- Problem 3: With the new tdata list you created, calculate the log2(tumor/normal) reads ratio for each line
- Problem 4: In a new file, 'logratio.txt', write each line's chromosome, position, and reads ratio. Your new file should have three lines.