Announcements

- **Assignments!**
  - Assignment 6 due tonight.
  - Assignment 7 out today, due next Friday at 11:59 p.m.
    - To help mitigate end-of-quarter stress, we'll allow you to submit until Sunday at 11:59 p.m. without penalty.
    - You can still use two late days beyond that.
- **Today's Agenda**
  - Discuss the MapReduce Programming Model, what a mapper is, what a reducer is, and how they can be chained together in a single pipeline of processes to analyze and process large data sets.
    - I'll present the map and reduce executables associated with the most canonical of MapReduce jobs: word counts. The slides present code in Python, since it's very short and easy to follow (even if you don't know Python). My lecture won't focus on the code, but rather on the general idea (which is fairly straightforward, imo.)
    - I'll discuss how very large data sets can be partitioned into many, many chunk files and processed by a large number of simultaneously executing map and reduce jobs on hundreds or even thousands of machines.
    - I'll discuss the group-by-key algorithm (very straightforward, actually) which is run on the full accumulation of mapper output files to generate the full set of reducer input files.
- **Next Week**
  - Additional Topics: Non-blocking I/O and event-driven programming (**`epoll`**, **`kqueue`**, and **`libev`**/**`libuv`** packages)

Code

- **The mapper!**
  - A mapper (or map executable) is a program that reads in an arbitrary data file and outputs a file of key-value pairs, one per line.
  - Here's an example of a Python program that reads in an arbitrary text file and outputs lines of the form " 1".

```
#!/usr/bin/env python
import sys
import re
import string

pattern = re.compile("^[a-z]+$") # matches purely alphabetic words
for line in sys.stdin:
    line = line.strip()
    tokens = line.split()
    for token in tokens:
        lowercaseword = token.lower()
        if pattern.match(lowercaseword):
            print '%s 1' % lowercaseword
```

  - The above program can be invoked as follows:

```
myth22> cat long-novels/anna-karenina.txt | mapper.py
```

  - Doing do will produce the following (condensed) output:

```
anna 1
karenina 1
by 1
leo 1
tolstoy 1
...
i 1
have 1
the 1
power 1
to 1
put 1
into 1
```

Code: Reducer

- **The reducer!**
  - A reducer is a second program that expects a sorted input file, where each line is a key/vector-of-values pair. (Some reducers, like the one I present here, actually does value-grouping by key as part of its execution).

```
#!/usr/bin/env python

from itertools import groupby
from operator import itemgetter
import sys

def read_mapper_output(file):
    for line in file:
        yield line.strip().split(' ')

def main():
    data = read_mapper_output(sys.stdin)
    for key, keygroup in groupby(data, itemgetter(0)):
        count = sum(int(v) for k, v in keygroup)
        print "%s %d" % (key, count)

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

  - The above reducer could be fed the sorted output of the previously supplied mapper if this chain of piped executables is supplied on the command line:

```
myth22> more long-novels/anna-karenina.txt | mapper.py | sort | reducer.py
```

  - The above chain of piped exeucutables would produce this:

```
a 6069
abandon 6
abandoned 9
abandonment 1
abashed 2
abasing 1
aber 1
abilities 1
...
zaraisky 4
zeal 3
zealously 1
zest 1
zhivahov 1
zigzag 1
zoological 2
zoology 1
zu 1
```