**Exercise 5**

*Get started with Apache Spark and Python*

**Prior Knowledge**

Unix Command Line Shell

Simple Python

**Learning Objectives**

Understand the Spark system

Use the Spark Python shell to interactively work with data

Submit Spark jobs locally and using YARN

Write SparkSQL code in Python  
WordCount!

**Software Requirements**

(see separate document for installation of these)

* Apache Spark 1.5.1
* Python 2.7.x
* Nano text editor or other text editor

**Part A. Spark Python Shell (pySpark)**

1. We are going to do a wordcount against a set of books downloaded from Project Gutenberg. Wordcount is the definitive Big Data program (sort of Hello World for Big Data) and its frankly embarrassing that we haven’t done one yet.
2. Apache Spark has a useful Python shell, which we can use to interactively test and run code. Since we have our data in HDFS, *we need to ensure HDFS is running.* (Follow the instructions from the Hadoop lab).
3. Let’s load some books into HDFS. In a terminal window (Ctrl-Alt-T)

hadoop fs -mkdir /user/oxclo/books

hadoop fs -put ~/datafiles/books/\* /user/oxclo/books/

1. Now, change to the Spark directory:  
   cd ~/spark-1.5.1
2. Now start the Spark Python command line tool – pyspark  
   bin/pyspark  
   1. You should see a lot of log come up, ending in something like:

15/10/25 23:39:52 INFO BlockManagerMaster: Registered BlockManager

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 1.5.1

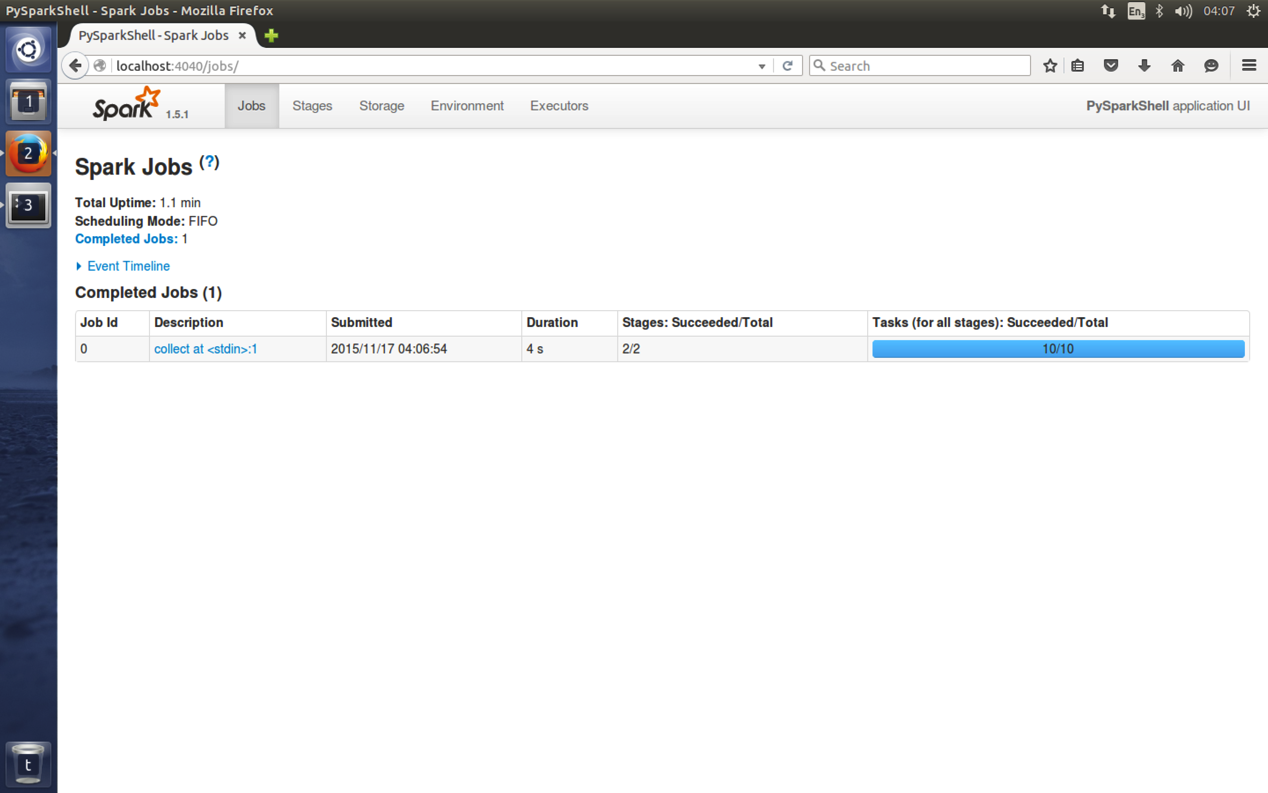
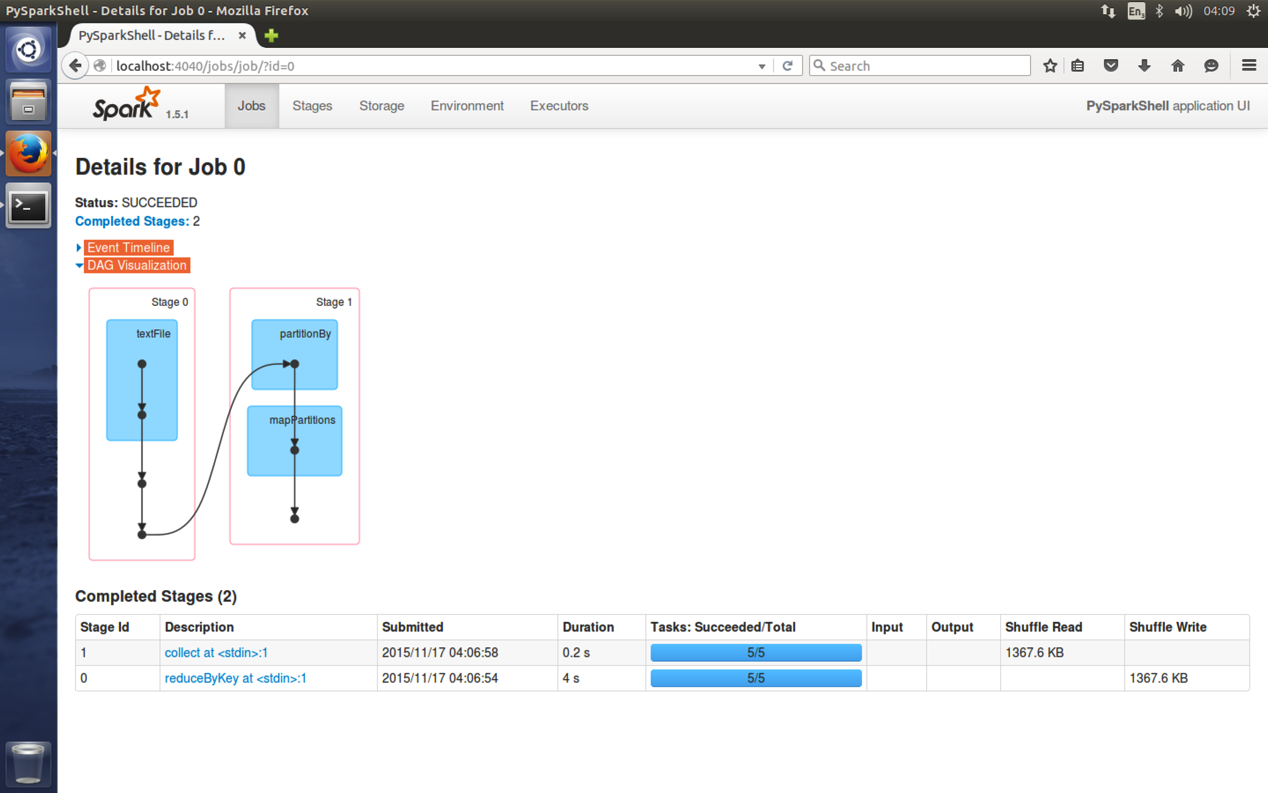
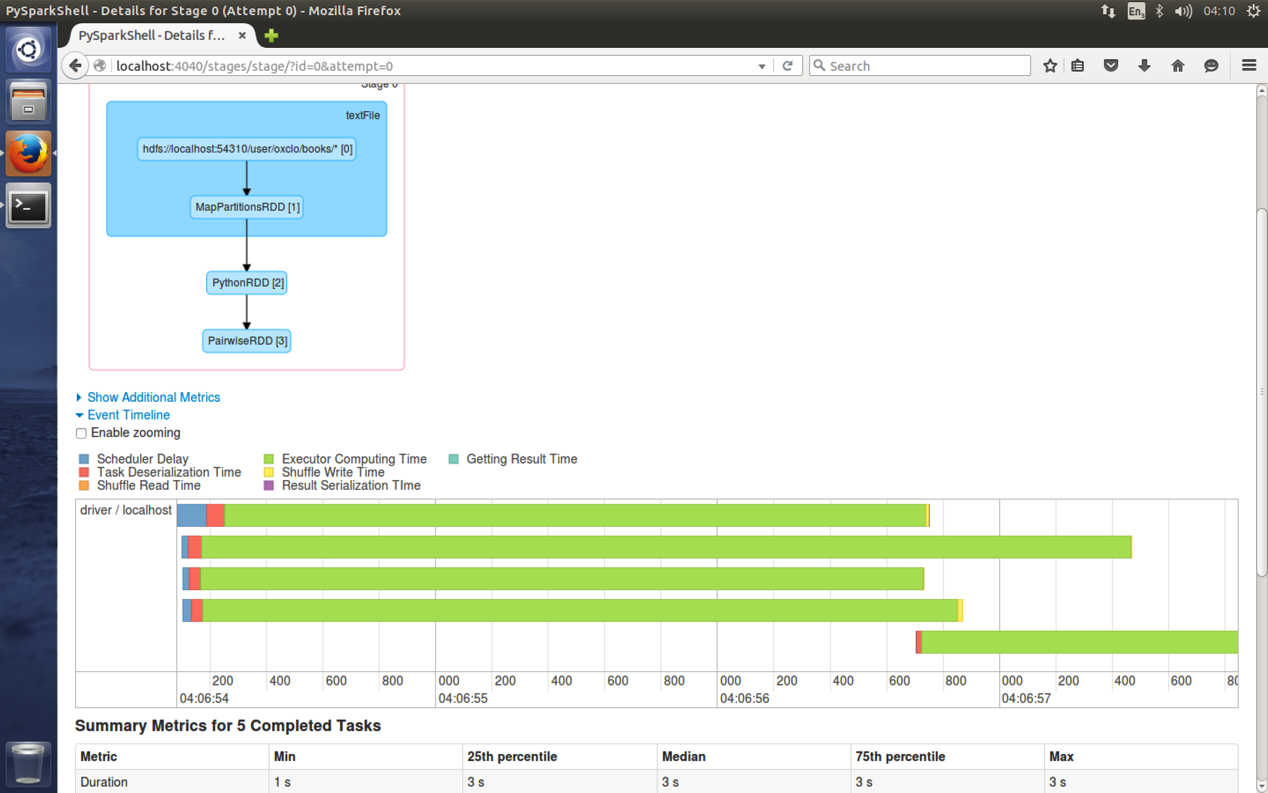
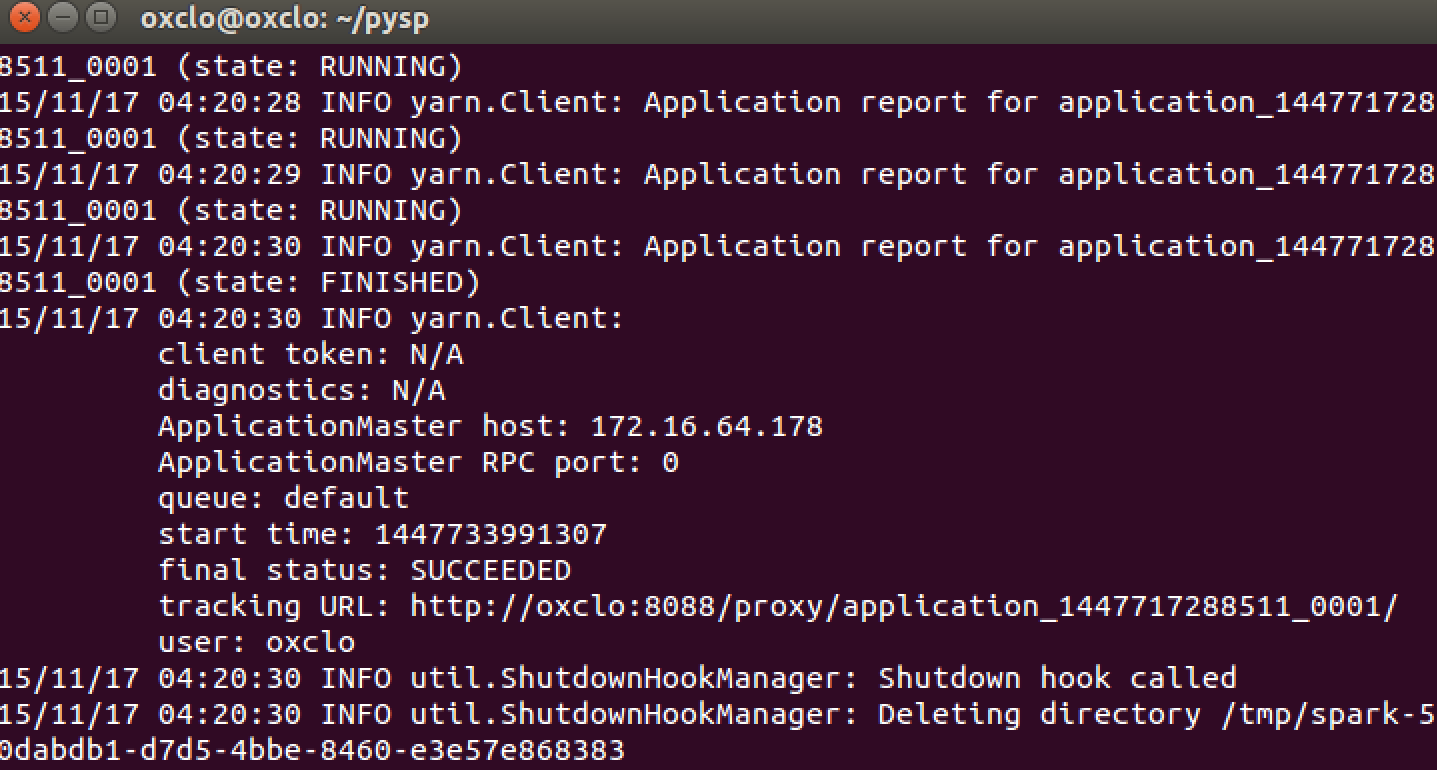
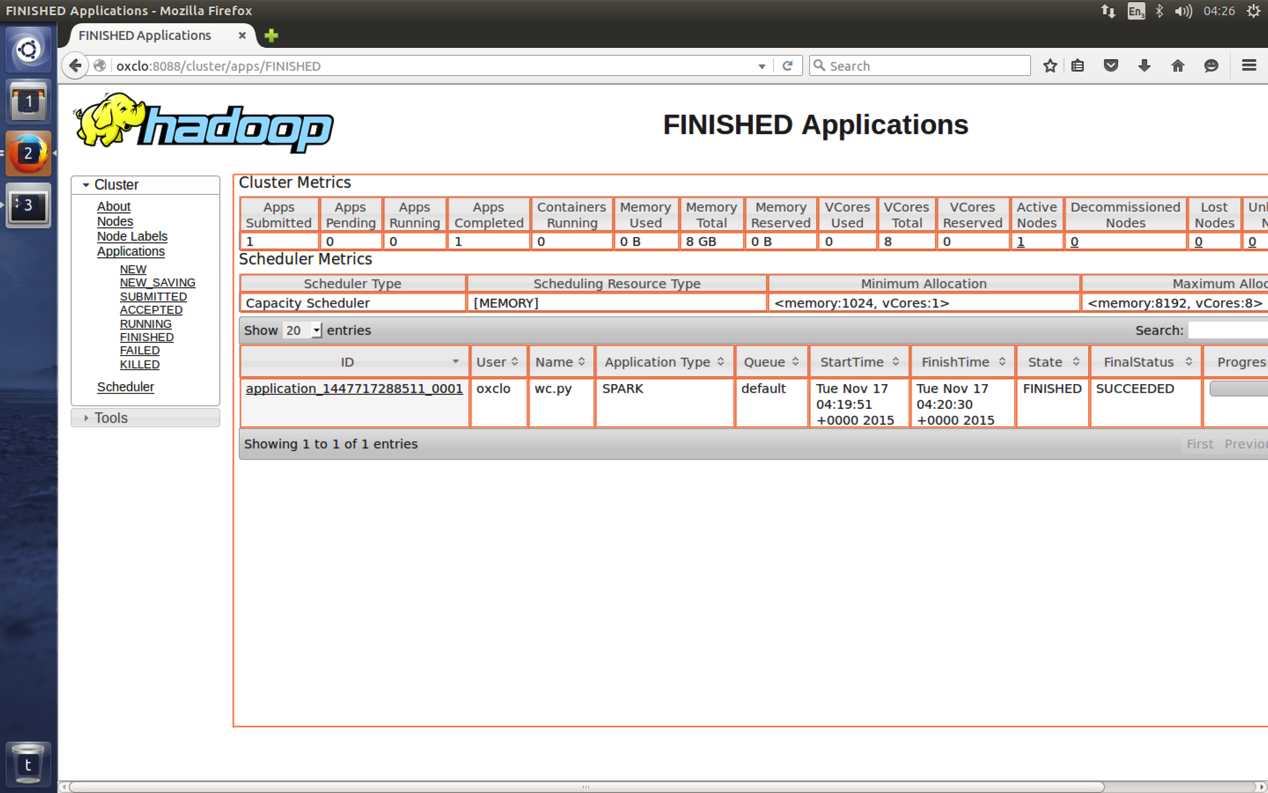
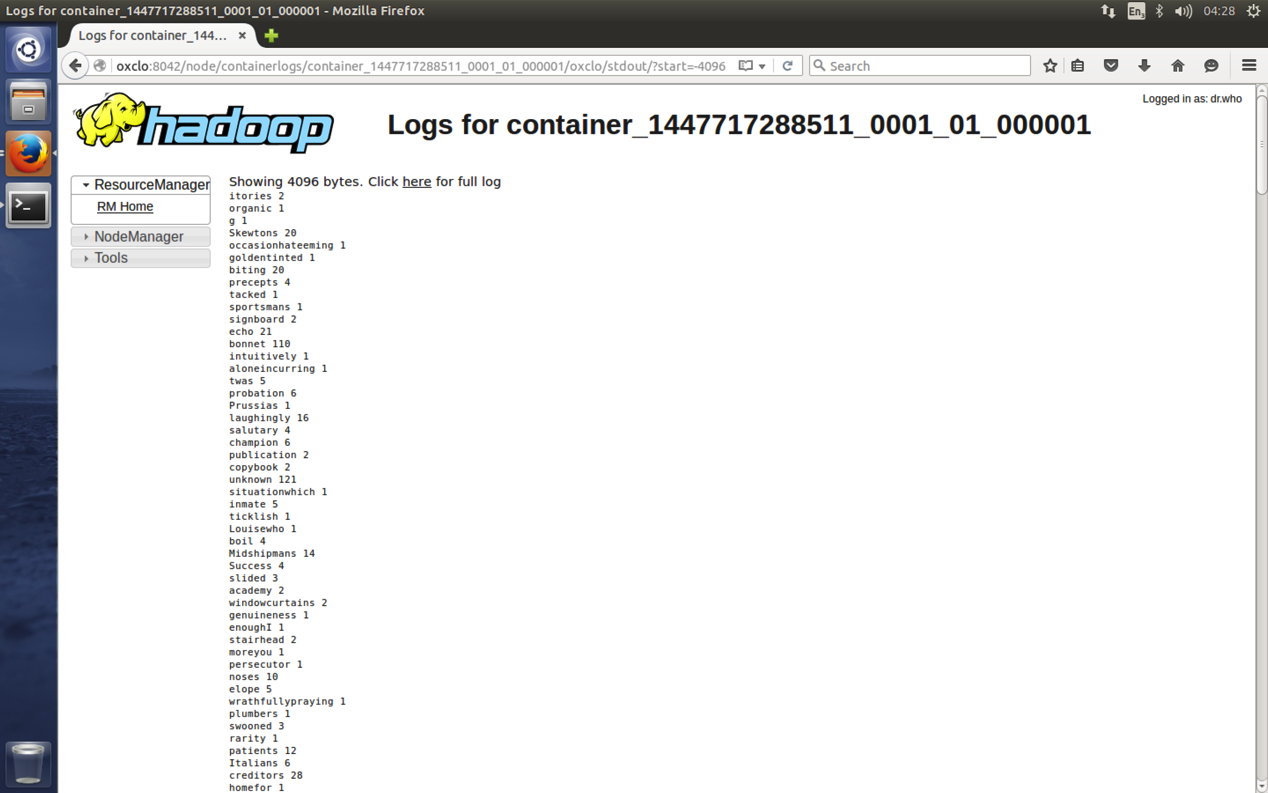
/\_/

Using Python version 2.7.6 (default, Jun 22 2015 17:58:13)

SparkContext available as sc, SQLContext available as sqlContext.

1. Now let’s load some data. We already have a SparkContext object defined in the shell (in a program you need to define one, which we will see later)
2. Unfortunately some of the input is handled as Unicode by Python and we want to get rid of that. So let’s start:  
     
   import unicodedata
3. Then type (on one line):  
     
   def u2a(u): return str(unicodedata.normalize('NFKD',u).  
   encode('ascii','ignore'))
4. We also want to remove any non-alphanumeric characters:  
   def strip(s): return ''.join(filter(str.isalpha, s))
5. Now we would like to load the books from HDFS. Now let’s load some data. We already have a SparkContext object defined in the shell (in a program you need to define one, which we will see later)  
     
   books = sc.textFile("hdfs://localhost:54310/user/oxclo/books/\*")
6. Let’s split the lines into words:  
     
   split = books.flatMap(lambda line: line.split())
7. Now let’s transform from Unicode to ascii   
   asc = split.map(u2a)
8. And remove non-alpha characters

stripped = asc.map(strip)

1. And we should put everything to lower case while we are cleaning it up  
     
   lower = stripped.map(lambda a: a.lower())
2. Finally we are ready to do the classic “WordCount” Map Reduce.   
   We first create a simple <K,V> pair of <word, count>. In the map phase, the count is always 1, since we haven’t yet reduced this.  
     
   numbered = lower.map(lambda word: (word, 1))
3. Next we reduce by adding the counts together for the same words:  
     
   wordcount = numbered.reduceByKey(lambda a,b: a+b)
4. Finally, we need to collect the results and print them. In Spark, they may be distributed across different RDD partitions on different machines, so the collect() method brings them together.  
     
   for k,v in wordcount.collect(): print k,v
5. You should see a lot of word counts go flying past.
6. Congratulations!
7. While the pyspark is still running browse to <http://localhost:4040>
8. You will see the Spark web console:  
   
9. Click on the blue link “collect at stdin”  
   This shows you how Spark converted your code into stages:  
   
10. From the stages section you can dig deeper into your execution plan:  
    
11. Quit the pyspark shell by typing quit()
12. Now let’s run the same code as a “job” instead of interactively.
13. Make a directory for your spark python code:  
    mkdir ~/pysp  
    cd ~/pysp
14. From <http://freo.me/oxclo-wc-py> copy the code into a file wc.py
15. Now configure the correct setup so Spark can find the Yarn system:  
      
    export HADOOP\_CONF\_DIR=/usr/local/hadoop/etc/hadoop/
16. Now submit the job via YARN:  
    ~/spark-1.5.1/bin/spark-submit --master yarn-cluster wc.py \ "hdfs://localhost:54310/user/oxclo/books/\*"
17. You can see the status of the job via the YARN webpage:  
    <http://localhost:8088>
18. You should also see a success via the command line:  
    
19. In the YARN web console, click on the FINISHED link in the left hand menu. You should see:  
    
20. Click on the application link and then click on the Logs link.
21. You can now find the stdout log from your code which has the result of the MR job: 
22. You can also run jobs locally on a single node without using YARN:  
    The local[\*] indicates to use as many threads as you have cores on your system:  
    ~/spark-1.5.1/bin/spark-submit --master local[\*] wc.py "hdfs://localhost:54310/user/oxclo/books/\*"
23. You could also use Spark’s own cluster manager or Apache Mesos as other options if you have those set up.
24. Congratulations, the lab is complete!