**Lab 10 – Big Data**

Computing and Informatics Design Lab

**Due:**

Posted in LEARN or in the Syllabus.

**Pre-Lab:**

Watch this video: <https://www.youtube.com/watch?v=D_wBRB5RPoY> on jsmapreduce.

Read Sections 2.6.1, 2.6.2, and 2.6.3 of the WebMapReduce User Guide: <http://webmapreduce.sourceforge.net/docs/using/interface.html#example-job>.

**Part 1: Freely Available Data Sets**

For each exercise, you can copy the output data to the following sites to make it more readable.

* <http://jsonprettyprint.com/>
* <http://xmlprettyprint.com/>
* <http://marianoguerra.github.io/json.human.js/>

Answer the questions for each exercise.

**US demographics:**

**Using data.gov, search for NCHS.**

### [NCHS - Leading Causes of Death: United States](https://catalog.data.gov/dataset/age-adjusted-death-rates-for-the-top-10-leading-causes-of-death-united-states-2013)

*U.S. Department of Health & Human Services —*

This dataset presents the age-adjusted death rates for the 10 leading causes of death in the United States beginning in 1999.

Open this data set in JSON format and look through it before answering the following question.

**Question 1:** Describe the layout of the JSON data and how it was organized.

Now open the same data set in XML format and look through it before answering the following question.

**Question 2:** Describe the layout of the XML data and how it was organized.

What is the number of accidental deaths in 2015 in Alabama and Alaska. You can use whichever response format you prefer.

**Question 3:** Report your findings. Why do you think these numbers are so different?

**Use the navigation tools in Data.gov to find a data set that interests you.**

**Question 4:** Howdid you navigate to that particular data set? What was that data set? Which format do you prefer and why?

**What to Turn In:**

Submit your answers for each portion of the lab in your answer sheet document.

**Part 2: Programming Exercise:**

**Introduction to JSMapReduce**:

JSMapReduce is no longer available online at its usual website, but some of its functionality still exists thanks to the Internet Archive’s Way Back Machine. You can access JSMapReduce at <https://web.archive.org/web/20141009181352/http://jsmapreduce.com/>. Note that some functionality will not work via the Internet Archive version of this site; however, fortunately, the example we plan to observe does work.

Using JSMapReduce, look through the first provided example under “Sample 1,” to better understand the map reduce process. Run it by clicking “Run” under “in-browser execution,” and observe the results in the “Job Status and Debug” window and “Output” window. Click “Reset” and use the “Single Step” button to trace through the behavior of the mappers and reducers.

Notice that, when clicking “Single Step,” a “mapper” is created. That is, the mapper function is called with only a small piece of the data.

**Question 5**: What, **in your own words** (don’t just paste the output!) does a mapper emit when you click on the “Single Step” button on JSMapReduce?

**Question 6**: What is each mapper use as its “piece” of the data? For example, is it a word, a paragraph, the whole poem, *etc.*?

**Question 7**: It seems surprising that this program calls so many mappers, each with a small part of the data. Why do you think MapReduce jobs are executed this way? For a small poem such as the one in this example, it would seem faster to just run the function on the whole poem at once. Why do you think this would run faster as the input becomes larger?

**Question 8**: Modify the line of code **var words\_list = data.split(' ');** by removing the space inside the parenthesis, and run your MapReduce job again (you may have to click Reset in between runs). What happens?

**Question 9**: Looking at the mapper function, how do you think the function takes its piece of the data, and produces its emitted values (*i.e.*, “Hope: 1”)? Describe what the code in the mapper function does, step-by-step.

**Question 10**: What does each reducer use as its input, and what does each reducer emit? See the Debug Log to the right when you run your MapReduce job for clues.

**Introduction to WebMapReduce:**

We will use a variant of JSMapReduce called WebMapReduce for the second part of this lab, since JSMapReduce is no longer fully functional on its original website. You can access WebMapReduce at [www.cs.drexel.edu/~wmm24/teaching/wmr.html](http://www.cs.drexel.edu/~wmm24/teaching/wmr.html), and log in with the credentials below:  
  
username: ci101   
password: ci101drexel

Let’s run some examples to see how it works.

First, create a job by clicking “New Job” on the left-hand menu (you should be there already once you log in for the first time). Give your job a name (the class is sharing this account, so try using your name or something else that will make your job name unique). The language will be Python. The Map Tasks and Reduce Tasks do not matter in this example, so you can enter a 1 for each of those, and alphabetic sort would be fine (numeric is also an option if you know your data will contain numbers).

Under Input, click the “Direct Input” button. If you want to upload a text file, you can click “Upload” here instead. Type/copy in some test input. I’ll use the same input from JSMapReduce, but feel free to be creative here!

hope is the thing with feathers

that perches in the soul

and sings the tune without the words

and never stops at all

and sweetest in the gale is heard

and sore must be the storm

that could abash the little bird

that keeps so many warm

Ive heard it in the chillest land

and on the strangest sea

yet never in extremity

it asked a crumb of me

Under Mapper and Reducer, again, you can use “Direct Input” to type in the code for the mapper and reducer, or you can save this to a text file and use the “Upload” option instead. Thanks to the WebMapReduce User Guide (<http://webmapreduce.sourceforge.net/docs/using/python.html>), we have the code for the same mapper and reducer behavior we saw on JSMapReduce to count the number of words that appear. (note – in Python, the indentation spacing matters!)

**def** reducer(key, values):

count = 0

**for** value **in** values:

count += int(value)

Wmr.emit(key, count)

**def** mapper(key, value):

words = key.split()

**for** word **in** words:

Wmr.emit(word, 1)

To run the job, click “Test” (do not click the other buttons “Submit” or “Save”).

**Question 11**: Unlike the mapper we used in the JSMapReduce example, this mapper emits a 1 for each word it finds, and it doesn’t bother counting them. Why does this approach still work?

**A Second Example: Computing Class Grade Averages**

Using the data below, modify the Python reducer in the example to compute the final grade for each student. Assume each individual grade carries an equal weight.

The syntax is slightly different from JavaScript, but using Python here will allow JSMapReduce to convert the data below from JSON (a common text format used to transmit data using web services) into an “associative array” or “dictionary” that you can access as follows:

{‘Alice’:95, ‘Bob’:66, ‘Carol’:57, ‘Dave’:81, ‘Eve’:86}

{‘Eve’:55, ‘Alice’:87, ‘Bob’:82, ‘Carol’:74, ‘Dave’:74}

{‘Dave’:79, ‘Bob’:56, ‘Carol’:70, ‘Eve’:50, ‘Alice’:63}

Here, each row might represent a test or assignment, and each student’s grade on that test/assignment appears in the row. Notice that the names don’t have to appear in the same order every time. They often do, but we’ll ask our Mapper to organize them by name just like our word-count mappers organized their input by each word, and then shuffled the counts so that each reducer got all the counts from each of those words at the end. You can have as many rows as you like – depending on how many mappers the MapReduce engine creates (this depends on how many computers are available to process your data), each mapper gets a certain number of rows to process. Think of it like dividing up a project among your friends; each one might do a certain (hopefully equal!) amount of the work.

Here is the mapper and the reducer to compute the average.

def mapper(key, value):

grade\_map = eval(key) # automatically parses the data in JSON format

# grade\_map['Alice'] contains 95 87 or 63 (and so on) depending on which row the mapper is reading

for student in grade\_map: # student is 'Alice', 'Bob', and so on

# get each of the 4 grades in your mapper data line

grade = grade\_map[student]

Wmr.emit(student, grade)

def reducer(key, values):

sum = 0

count = 0

for value in values:

sum = sum + float(value)

count = count + 1

if count > 0:

average = sum / count

Wmr.emit(key, average)

Make up your own grade data in JSMapReduce, using the format above, create a new WebMapReduce job like you did in the previous example, and compute each student’s average grade.

**Question 12:** What output do you get from WebMapReduce?

**Processing Philadelphia School Enrollment Data with WebMapReduce:**

Using the provided data, modify the Python mapper from the example to compute the total number of enrolled students in Philadelphia schools and the total number of student dropouts.

{ ‘Name’:’ARISE Academy Charter High School’, ‘Type’:’CS’, ‘Enrollments’:’183’, ‘Male Dropouts’:’1’, ‘Female Dropouts’:’1’, ‘Dropouts’:’2’ }

{ ‘Name’:’ASPIRA Bilingual Cyber Charter School’, ‘Type’:’CS’, ‘Enrollments’:’57’, ‘Male Dropouts’:’2’, ‘Female Dropouts’:’6’, ‘Dropouts’:’8’ }

{ ‘Name’:’Ad Prima CS’, ‘Type’:’CS’, ‘Enrollments’:’26’, ‘Male Dropouts’:’0’, ‘Female Dropouts’:’0’, ‘Dropouts’:’0’ }

{ ‘Name’:’Alliance for Progress CS’, ‘Type’:’CS’, ‘Enrollments’:’24’, ‘Male Dropouts’:’0’, ‘Female Dropouts’:’0’, ‘Dropouts’:’0’ }

{ ‘Name’:’Philadelphia City SD’, ‘Type’:’SD’, ‘Enrollments’:’63983’, ‘Male Dropouts’:’3092’, ‘Female Dropouts’:’2644’, ‘Dropouts’:’5736’ }

There are more fields provided than you require. We should filter these out. You can use an **if** statement to ask whether or not an entry is a certain value. For example, if we only wanted to count Alice’s grades, we could modify the mapper in the previous example as follows (replacing the grade = grade\_map[student] and Wmr.emit() lines with the if statement below):

**if student ==** 'Alice'**:  
 # only runs Alice’s grades**

**grade = grade\_map[student]**

**Wmr.emit(student, grade)**

**# now that we’re not indented**

**# code down here runs no matter what**

**A Few Thoughts about this Assignment:**

This exercise is very similar to what we did in JSMapReduce, but there will be a few changes with the switch to WebMapReduce, which we describe here:

1. WebMapReduce uses **key**, **value** instead of **jsmr\_context** and **data** for the mapper parameters, and **key**, **values** for the reducer parameters.
2. **jsmr\_context.Emit()** becomes **Wmr.emit()**.
3. The **Mapper** and **Reducer** function names are lower-case in WebMapReduce.
4. Finally, the **Reducer** can emit **(key, sum)** instead of a single string as in the reducer example below.

**What to Turn In:**

**Question 13:** Each student must submit code for the mapper, reducer, and your results for each portion of the lab.