**Question 1**:

It shows the word count of the words in first line of the poem.

**Question 2**:

The mapper uses the first line of the poem as the data. It then goes line by line

**Question 3**:

MapReduce is run in this way so that when calculating larger inputs you can minimize heat output and time needed to run. For example, if you are going off of a 30 page document, or something larger, for one computer to go line by line to add together the word counts it would take substantially longer than if several computers just took a small portion of the lines.

**Question 4**:

It changed from counting the words to counting the letters, since there is nothing in the code to group the letters together by using spaces as separators.

**Question 5**:

The code groups together the letters based off of the spaces. The code then inputs a 1 for that word. It then groups together like words and then adds together the total “1’s” for that word. It then outputs the total word count for that word.

**Question 6**:

By using *var count = word\_count\_map[word]* it is able to count the amount of times a word is repeated and emit the word and the count.

**Question 7**:

This approach still works because it is adding the word count at the end instead of grouping together the same types of words in each line and then adding each line together.

**Question 8:** What output do you get from WebMapReduce?

From WebMapReduce you get the average of each student.

**Question 9:** Paste your code for the mapper and reducer below:

def mapper(key, value):

grade\_map = eval(key)

for student in grade\_map:

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:

total = sum

Wmr.emit(key, total)

Output

Dropouts 5746.0

Enrollments 64273.0

**What to Turn In:**

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