# Feedback — Experimenting with MapReduce using JSMapreduce

You submitted this quiz on **Sun 2 Jun 2013 10:37 AM PDT (UTC -0700)**. You got a score of **10.00** out of **10.00**.

## Experimenting with MapReduce using JSMapreduce

In this assignment, you'll be using JSMapreduce at http://jsmapreduce.com .

JSMapreduce is a cloud-based Mapreduce service that allows you to single-step a MapReduce job and see what's going on at each stage.

Please register for a free account on jsmapreduce.com, and make sure to always be logged in when using it for this course. It works best in the Chrome browser.

This assignment involves following the steps below to explore the features of jsmapreduce. The last problem will involve

**Step 1: Register for a free account on jsmapreduce.com**

Once you've registered, make sure you're logged in.

(You do not need to purchase any credits for this exercise.)

**Step 2: Select Sample 3 on the main page.**

This shows you a sample mapreduce job written in Python.

**Step 3: Find the Single Step button, and single-step through the job.**

Observe that lines of JSON records from the "INPUT data" window are fed one by one to the Mapper() function, causing the Mapper() to emit key-value pairs.

Once all the input data is consumed, observe the grouped key-value pairs being fed to the Reducer() function, which outputs final counts.

Make sure you understand all the output in the "Job Status and Debug" window, as you single-step through Sample 3.

You have executed a mapreduce on a single machine. For a very large set of input data, this could have been run on many machines.

**Step 4: Watch the JSMapreduce intro video**

http://www.youtube.com/watch?v=i-Uj9rp_sPA

**Step 5: Now make some changes to the input data.**

Click 'Reset'

In the "INPUT data" window, add more JSON records, with counts for things besides Apples/Bananas/Cherries -- Dates, Figs, and Grapes; or whatever you like.

Click 'save' to save your data.

Single-step through your new data, watching the debug output in the "Job Status and Debug" window.

Now click "Reset", and then "Run" to run through the entire job.

**Step 6: Now make changes to the Mapper function.**

Click 'Reset'.

In the "Mapper function" window, modify the code to generate different key-value pairs.

For example, instead of e.g. key="Apples", value="4", you might change it to count occurrences of letters in the fruit name (e.g. key="p", value="2", ...).

Click 'save' to save your code.

Single-step through your code, and make sure you understand everything in the "Job Status and Debug" window.

Now "Reset" and "Run" your code at full speed.

Several things to note:

(a) The Mapper() function is called once per input record.

(b) A single Mapper() call may result in many key-value pairs being emitted. It depends on the problem.
For some input records, the Mapper may emit no key-value pairs, if there is nothing of interest in the input.

(c) The choice of keys are values can be anything -- it depends completely on the problem you're trying to solve.

(d) The output key and value must always be strings. If value is a number, it must be "stringized."
This is true of most mapreduce implementations (mapreduce is generally not type-safe!).

(e) [This is IMPORTANT] Although the keys and values are strings, they may be arbitrarily complex records, encoded as serialized JSON.
For example, for web-related input records, the key might be a domain name, and the value might be some complex JSON structure describing a user's interaction with the site. The Reducers would then sum or otherwise combine the site-visit data, one domain at a time.

**Step 7: Now make changes to the Reducer function.**

Several things to note first:

(a) A single Reducer() call receives all key-value pairs with the same key.

(b) The scaling of the Reduce phase is limited by the variety of keys emitted by the

Mapper phase.
If the Mappers generate many key-value pairs, but the keys have little variety (e.g. only 10 different keys), then the Reduce phase has to do all its work on just a few machines; because ALL the key-value pairs with a single key have to be reduced together.
This can result in a "bad" mapreduce. A solution is to artificially create more variation in the keys, and then do the final reduction in a second mapreduce.

(c) The Reducer can output anything it wants. For most purposes, it outputs some sort of summation of the values for a single key.

(d) Often, an analytic job cannot be accomplished in a single Mapreduce.
In this case, the first Mapreduce should ouput data that is ingestible by the second Mapreduce.

*Ungraded Exercise for Step 7*

1. Sample 3's Reducer() output is in a human-readable form. Change it to output results in a more standard, more computable form.
2. Experiment with making the entire job more complex; and making the Reducer's output be formatted as a next stage Mapper's input.
3. Cut-and-paste your first Mapreduce's output into the INPUT box, and "save" it; now write another Mapper function to process it ...

**Step 8: Observe Sample 4 (Python Poker Analysis)**

Make sure you've watched the entire JSMapreduce intro video (see link above).

Single-step through a portion of Sample 4's Mapper phase. Make sure you understand the output in the "Job Status and Debug" window.

(Note that the first "step" is the "Kernel" execution, which generates the input data. You can ignore this step.)

Click "Reset" and "Run" to watch the entire job execute.
It will take a few minutes, since it is running on a single machine. If it were running on many machines, it could be instantaneous.

**Step 9: Modify Sample 4 (Python Poker Analysis)**

***In this step, you will turn in the answer to a problem***

Modify the Mapper function in Sample 4, to make it count a new type of (fake) poker hand: "4cardstraight" -- i.e. a straight, but with only 4 cards in a row instead of 5.

A hand is counted as a "4cardstraight" if it includes 4 cards in a row, but not 5 (so it's not a regular 5-card straight).
When a hand qualifies as multiple types, the Mapper() returns the highest-value hand; for this purpose, the value of "4cardstraight" will be just below a straight.

When the input data indicates a "4cardstraight", your modified Mapper() function should emit the new key "4cardstraight", with a stringized count of '1'.

If you did it correctly, the Reducer() function should output "4cardstraight":#, where '#' is the count of "4cardstraight"'s.
You should not need to modify the Reducer().

Save your modified Mapper() function, execute it, and turn in the output.

What to turn in: A file output.txt with a single number representing the number of 4cardstraights.

[Note: If you prefer Javascript to Python, you can instead modify Sample 2, the Javascript equivalent of Sample 4.
A single-machine Javascript job will execute entirely within your browser, so may also be more reliable if the server becomes stressed.]

## Question 1

How many "straighflush" hands are there in this dataset?

**You entered:**

40

| Your Answer | | Score | Explanation |
|---|---|---|---|
| 40 | ✓ | 2.00 | |
| Total | | 2.00 / 2.00 | |

## Question 2

How many occurrences of "4cardstraight" are there in this dataset?

**You entered:**

97476

| Your Answer | | Score | Explanation |
|---|---|---|---|
| 97476 | ✓ | 8.00 | |
| Total | | 8.00 / 8.00 | |

**Question Explanation**

See Step 9 in the material above for the definition of "4cardstraight"