CMPS 6610 Lab 03

Name (Team Member 1):__________________________________
Name (Team Member 2):__________________________________

In this lab you will practice using the `map` and `reduce` functions. These functions are commonly used together in a `map-reduce` framework, used by Google and others to parallelize and scale common computations.

Part 1: Counting Words

In the first part, we will use map-reduce to count how often each word appears in a sequence of documents. E.g. if the input is two documents:

```
['i am sam i am', 'sam is ham']
```

then the output should be

```
[('am', 2), ('ham', 1), ('i', 2), ('is', 1), ('sam', 2)]
```

We have given you the implementation of the main map-reduce logic

```
def run_map_reduce(map_f, reduce_f, docs)
```

To use this function to count words, you'll need to implement your own `map_f` and `reduce_f` functions, described below.

1. Complete `word_count_map` and test it with `test_word_count_map`.
2. Complete `word_count_reduce` and test it with `test_word_count_reduce`.
3. If the above are correct, then you should now be able to test it the full solution `test_word_count`
4. Assume that a word `w` appears `n` times. What is the **work** and **span** of `word_count_reduce` for this word, assuming a parallel implementation of the `reduce` function?

Enter answer here

5. Why are we going through all this trouble? Couldn't I just use this function to count words?

```
docs = ['i am sam i am', 'sam is ham']
counts = {}
for doc in docs:
    for term in doc.split():
        counts[term] = counts.get(term, 0) + 1
# counts = {'i': 2, 'am': 2, 'sam': 2, 'is': 1, 'ham': 1}
```

What is the problem that prevents us from easily parallelizing this solution?

Enter answer here

Part 2: Sentiment analysis

Finally, we'll adapt our approach above to perform a simple type of sentiment analysis. Given a document, rather than counting words, we will instead count the number of positive and negative terms in the document, given a predefined list of terms. E.g., if the input sentence is `it was a terrible waste of time` and the terms `terrible` and `waste` are in our list of negative terms, then the output is

```
[('negative', 1), ('negative', 1)]
```

6. Complete the `sentiment_map` function to implement the above idea and test it with `test_sentiment_map`.

7. Since the output here is similar to the word count problem, we will reuse `word_count_reduce` to compute the total number of positive and negative terms in a sequence of documents. Confirm your results work by running `test_sentiment`.