Pseudocode: Wordcount Mapper

```
function Map(doc d)
    for all word w ∈ doc d
        output(word w, count 1)
```

Python Code

```
function Map(doc d)
    for all word w ∈ doc d
        output(word w, count 1)
```

```
#!/usr/bin/env python
import sys
for line in sys.stdin:
    line = line.strip()
    words = line.split()
    for word in words:
        print('%s\t%s' % (word, 1))
```

get all lines from stdin

remove leading and trailing whitespace

split the line into words,
and get all words

Pseudocode: Wordcount Reducer

```
function Reduce(word w, counts[c1, c2, …])
    sum = 0
    for all count c ∈ counts[c1, c2, …]
        sum = sum + c
    output(word w, count sum)
# But this is for one-reduce-one-word case
# Remember that one reducer can receive multiple
keys (thus words)
# So we need Python dictionary
```

Pseudocode: Reducer (Cont.)

```
function Reduce([words, counts])
    wordcount_dic = {}
    for all w ∈ words
        if w already in wordcount_dic
            add count c to wordcount_dic[w]
        else
            wordcount_dic[w]=c
    for all keys in wordcount_dic
        output(key, wordcount_dic[key])
```

Python Code

```
function Reduce([words, counts])

    wordcount_dic = {}
    for all w ∈ words
```

```
#!/usr/bin/env python
import sys
word2count = {}
for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:
        continue
```

split when \t is found & max 1 split can be done.

parse input from mapper.py

convert count (currently a string) to int

Python Code (Cont.)

```
if w already in wordcount_dic
        add count c to wordcount_dic[w]
else
        wordcount_dic[w]=c
for all keys in wordcount_dic
        output(key,wordcount_dic[key])
```

```
    try:
        word2count[word] = word2count[word]+count
    except:
        word2count[word] = count
for word in word2count.keys():
    print('%s\t%s'% ( word, word2count[word] ))
```

Frequent Singletons Pseudocode

```
function Map(transactions t)
      for all items m ∈ transactions
            output(singleton m, count 1)

function Reduce([singletons, counts])
      itemcount_dic = {}
      support = n
      for all m ∈ singletons
            if m already in itemcount_dic
                  add count c to itemcount_dic[m]
            else
                  itemcount_dic[m]=c
      for all keys in itemcount_dic
            if itemcount_dic[key]>=support
                  output(key, itemcount_dic[key])
```

Frequent Doubletons Pseudocode

```
function Map(transactions t)
      for all items i ∈ transactions
            for all items j after i in the same transaction
                  output(doubleton (i,j), count 1)

function Reduce([doubletons, counts])
      itemcount_dic = {}
      support = n
      for all m ∈ doubletons
            if m already in itemcount_dic
                  add count c to itemcount_dic[m]
            else
                  itemcount_dic[m]=c
      for all keys in itemcount_dic
            if itemcount_dic[key]>=support
                  output(key, itemcount_dic[key])
```