```
data = [["hello world"] * 8] * 4
```

```
# data = a bunch of files
[['hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world'],
 ['hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world'],
 ['hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world'],
 ['hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world',
  'hello world']]
```

```
for input_split in data:
input_split =
```

```
#input_split = each file
['hello world',
 'hello world',
 'hello world',
 'hello world',
 'hello world',
 'hello world',
 'hello world',
 'hello world']
```

```
def mapper(record):
    """The Map part of "MapReduce"

    The mapper function maps input key/value pairs to a set of intermediate
    key/value pairs. A given input pair may map to zero or many output pairs,
    these are all consumed by the reducer function in the next step.
    """

    counts = defaultdict(int)  # has default initial value of 0 for counting
    for word in record.split():
        counts[word] = counts[word] + 1  # increment count for word

    return list(counts.items())  # return list of tuples only
```

Record = each line in the file
for record in input_split
record = hello world

mapper(record) = a list of tuples indicating each word and its count for each line

[('hello', 1), ('world', 1)]

```
def map_input_split(input_split):
    """Send input splits to the mapper function individually

    This is extra glue that we need to apply the map function in Python, you
    will not need to write this logic for a standard MapReduce job in Hadoop.
    """

    results = []
    for record in input_split:
        results.extend(mapper(record))  # apply mapper function to record

    return results
```

```
map_input_split(input_split)
```

map_input_split (input_split) = a list of tuples indicating each word and its count for each file.
[('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),

('hello', 1),
('world', 1),
('hello', 1),
('world', 1),
('hello', 1),
('world', 1),
('hello', 1),
('world', 1),
('hello', 1),
('world', 1),
('hello', 1),
('world', 1)]

```
mapped_result_splits = []
for input_split in data: # for each node in cluster
    mapped_result_splits.append(map_input_split(input_split))
```

```
# mapped_result_splits = a list of map_input_split(input_split)
[[('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1)],
 [('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
  ('hello', 1),
  ('world', 1),
```

```
 ('hello', 1),
 ('world', 1)],
[('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1)],
[('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1),
 ('hello', 1),
 ('world', 1)]]
```

```
def group_by_key(mapped_result_split):
    """Grouped mapped result split by key
    """

    groups = defaultdict(list)  # has default initial value of [] for appending
    for (k, v) in mapped_result_split:
        groups[k].append((k, v))  # group key value pairs by key

    return list(groups.values())  # return list of grouped key value pairs only
```

```
mapped_result_split = map_input_split(input_split)
```

```
# grouped_result_split = group_by_key(mapped_result_split)
[[('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1)],
 [('world', 1),
  ('world', 1),
  ('world', 1),
  ('world', 1),
  ('world', 1),
  ('world', 1),
  ('world', 1),
  ('world', 1)]]
```

```
def reduce_grouped_result_split(grouped_result_split):
    """Reduce grouped result split by applying the reducer function iteratively
    """

    reduced_result_split = []
    for group in grouped_result_split:
        reduced_group = reduce(reducer, group)  # apply reducer function iteratively
        reduced_result_split.append(reduced_group)

    return reduced_result_split
```

```
#group =
 [('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1),
  ('hello', 1)]
```

```
def reducer(x, y):
    """The Reduce part of "MapReduce"
```

```
    The reducer function reduces a set of intermediate values which share a key
    to a smaller set of values.
    """

    return (x[0], x[1] + y[1])  # same key, add value
```

iterator in group = (‘word’, 1)
reduced_group = reduce(reducer, group)
('hello', 8)
('world', 8)

```
reduce_grouped_result_split(group_by_key(mapped_result_split))
```

reduce_grouped_result_split(grouped_result_split) = for each file
[('hello', 8), ('world', 8)]

```
reduced_result_splits = []
for mapped_result_split in mapped_result_splits:
    reduced_result_splits.append(reduce_grouped_result_split(group_by_key(mapped_result_split)))
```

reduced_result_splits: for the whole chunk
[[('hello', 8), ('world', 8)],
 [('hello', 8), ('world', 8)],
 [('hello', 8), ('world', 8)],
 [('hello', 8), ('world', 8)]]

```
def combine_reduced_result_splits(reduced_result_splits):
    """Combine the reduced result splits
    """

    results = {}
    for reduced_result_split in reduced_result_splits:
        for (k, v) in reduced_result_split:
            if k not in results:
                results[k] = (k, v)
            else:
                results[k] = reducer(results[k], (k, v))

    return list(results.values())  # return list of combined key value pairs only
```

```
results = combine_reduced_result_splits(reduced_result_splits)
```

results: add together
[('hello', 32), ('world', 32)]

```
for (word, count) in results:
    print("{}: {}".format(word, count))
```

#Output
hello: 32
world: 32